# Programming in Java

2020-2021

المرحلة الثالثة/ الفصل الدراسي الاول

استاذ المادة

أ.م. د. سراب  مجيد حميد

## Java programming language

Java programming language was originally developed by Sun Microsystems, which was initiated by James Gosling and released in 1995 as core component of Sun Microsystems' Java platform. It is owned by Oracle. It is used for:

- Mobile applications.
- Desktop applications
- Web applications
- Web servers and application servers
- Games
- Database connection
- And much, much more!

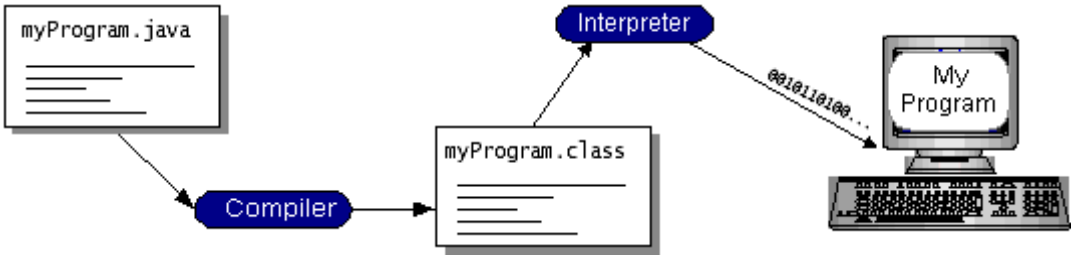
## Significant Language Features

- **Platform Independence** - Java compilers do not produce native object code for a particular platform but rather 'byte code' instructions for the Java Virtual Machine (JVM). Making Java code work on a particular platform is then simply a matter of writing a byte code interpreter to simulate a JVM. What this all means is that the same compiled byte code will run unmodified on any platform that supports Java.
- **Object Orientation** - Java is a pure object-oriented language. This means that everything in a Java program is an object and everything is descended from a root object class.
- **Rich Standard Library** - One of Java's most attractive features is its standard library. The Java environment includes hundreds of classes and methods in six major functional areas.
    - *Language Support* classes for advanced language features such as strings, arrays, threads, and exception handling.
    - *Utility* classes like a random number generator, date and time functions, and container classes.
    - *Input/output* classes to read and write data of many types to and from a variety of sources.
    - *Networking* classes to allow inter-computer communications over a local network or the Internet.
    - *Abstract Window Toolkit* for creating platform-independent GUI applications.
    - *Applet* is a class that lets you create Java programs that can be downloaded and run on a client browser.

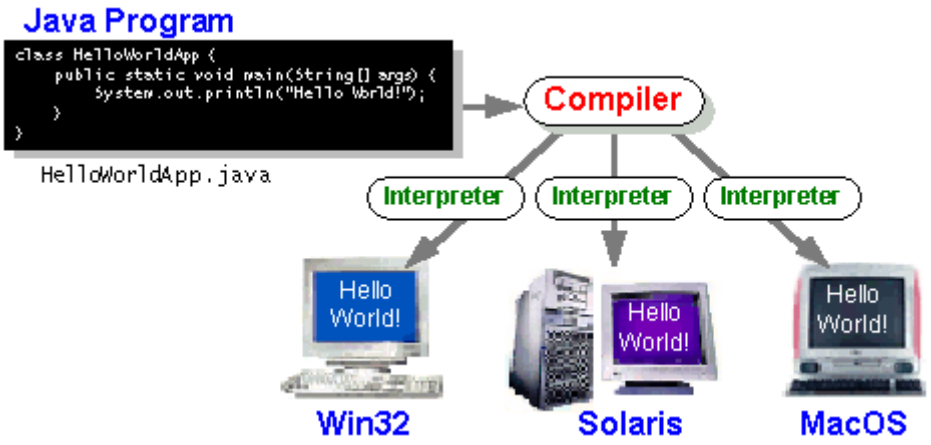
- **Applet Interface** - In addition to being able to create stand-alone applications, Java developers can create programs that can downloaded from a web page and run on a client browser.
- **Familiar C++-like Syntax** - One of the factors enabling the rapid adoption of Java is the similarity of the Java syntax to that of the popular C++ programming language.
- **Garbage Collection** - Java does not require programmers to explicitly free dynamically allocated memory. This makes Java programs easier to write and less prone to memory errors.

## Running Programs in Java

With most programming languages, you either compile or interpret a program so that you can run it on your computer. The Java programming language is unusual in that a program is both compiled and interpreted. With the compiler, first you translate a program into an intermediate language called **Java bytecodes**-the **platform-independent codes** interpreted by the interpreter on the Java platform. The interpreter parses and runs each Java bytecode instruction on the computer. Compilation happens just once; interpretation occurs each time the program is executed.




Java bytecodes help make "**write once, run anywhere**" possible. You can compile your program into bytecodes on any platform that has a Java compiler. The bytecodes can then be run on any implementation of the Java VM. That means that as long as a computer has a Java VM, the same program written in the Java programming language can run on Windows 2000, a Solaris workstation, or on an iMac. Figure 2 illustrates this.




## Java is Object-Oriented

Object oriented programming is the catch phrase of computer programming in the 1990's. Although object oriented programming has been around in one form or another since the Simula language was invented in the 1960's, it's really begun to take hold in modern GUI environments like Windows, Motif and the Mac. In object-oriented programs data is represented by objects. Objects have two sections, fields (instance variables) and methods. **Fields** tell you what an object is. **Methods** (the instructions of java that are used in writing a program) tell you what an object does. These fields and methods are closely tied to the object's real world characteristics and behavior. When a program is run, messages are passed back and forth between objects. When an object receives a message it responds accordingly as defined by its methods.

Object oriented programming is alleged to have a number of advantages including:

- Simpler, easier to read programs
- More efficient reuse of code
- Faster time to market
- More robust, error-free code

In practice object-oriented programs have been just as slow, expensive and buggy as traditional non-object-oriented programs. In large part this is because the most popular object-oriented language is C++. C++ is a complex, difficult language that shares all the obfuscation of C while sharing none of C's efficiencies. It is possible in practice to write clean, easy-to-read Java code. In C++ this is almost unheard of outside of programming textbooks.

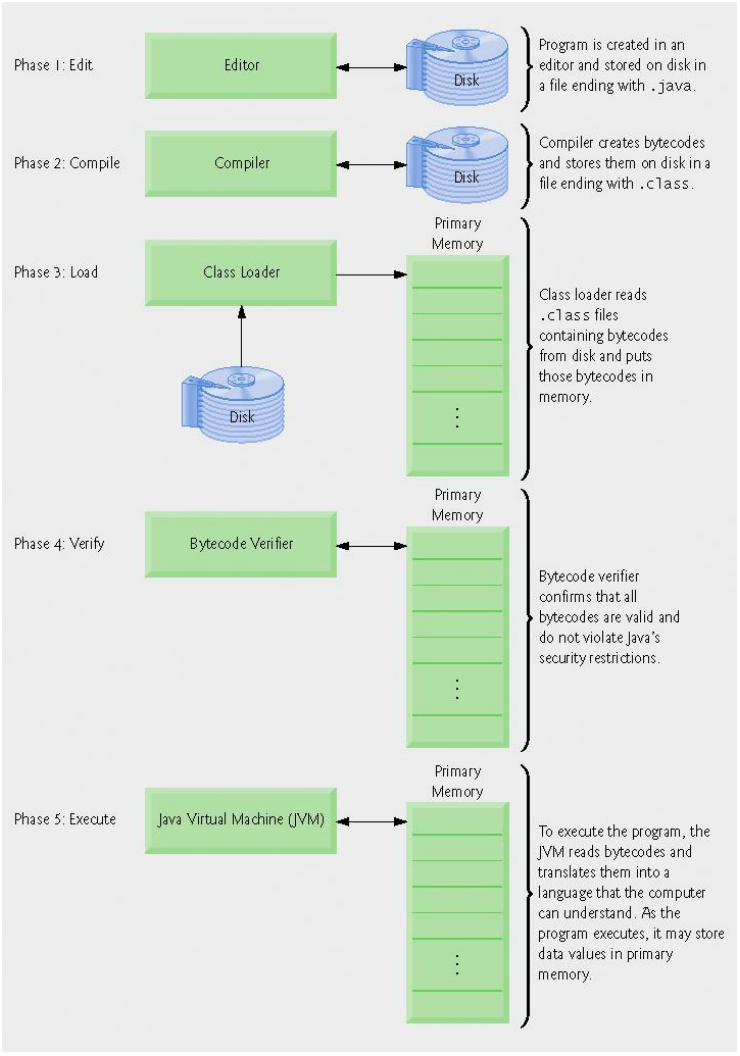
## Java Program Execution

In order to run a program in Java you have to install a Java Virtual Machine (Java Platform), there are 3 types of Java Platforms from Sun Microsystems:

- Java 2 Platform, Standard Edition (J2SE)
- Java 2 Platform, Enterprise Edition (J2EE)
- Java 2 Platform, Micro Edition (J2ME)

The Java 2 Platform, Enterprise Edition (J2EE) is geared toward developing large-scale, distributed networking applications and Web-based applications. The Java 2 Platform, Micro Edition (J2ME) is geared toward development of applications for small, memory-constrained devices, such as cell phones, pagers and PDAs, in these lectures we will discuss the basics of Java using the J2SE.

Java programs normally go through five phases to be executed as shown in the following figure.



## Types of Java Programs

There are two types of programs in Java:

- Applications
- Applets

An **application** is a program (such as a word-processor program, a spreadsheet program, a drawing program or an e-mail program) that normally is stored and executed from the user's local computer.

**Writing Simple Application**

In order to create an application in Java you have to do the following steps:

1- Create a source file (writing the code).

2- Compile the source file into a bytecode file.

3- Run the program (code) contained in the bytecode file.

**/* java application used to display the message "Welcome to Java programming!" on the standard output. */**

```
// Welcome1.java
// Text-printing program.

public class Welcome1
{
// main method begins execution of Java application
public static void main( String[] args )
{
System.out.println( "Welcome to Java Programming!" );
} // end method main
} // end class Welcome1
```

**Output:**

Welcome to Java Programming!

An **applet** is a small program that normally is stored on a remote computer that users connect to via a World Wide Web browser. The remote computer is known as a Web server. Applets are loaded from a remote computer into the browser, executed in the browser and discarded when execution completes. To execute an applet again, the

user must point a browser at the appropriate location on the World Wide Web and reload the program into the browser.

Web browsers such as Netscape or Microsoft Internet Explorer are used to view documents on the World Wide Web called Hypertext Markup Language (HTML) documents. HTML describes the format of a document in a manner that is understood by the browser application. An HTML document may contain a Java applet. When the browser sees an applet referenced in a HTML document, the browser lunches the Java class loader to load the applet (normally from the location where the HTML document is stored). Each browser that supports Java has a built-in Java interpreter (i.e. JVM). After the applet load, the browser's Java interpreter executes the applet.

## Java - Basic Syntax

When we consider a Java program it can be defined as a collection of objects that communicate via invoking each other's methods. Let us now briefly look into what do class, object, methods and instance variables mean.

- **Object :** Objects have states and behaviors

  **Example**: A dog has states: color, name, breed as well as behaviors:- wagging, barking, eating. An object is an instance of a class.
- **Class :** A class can be defined as a template/ blue print that describes the behaviors/states that object of its type support.
- **Methods:** A method is a behavior. A class can contain many methods. It is in methods where the logics are written, data is manipulated and all the actions are executed.
- **Instance Variables:** Each object has its unique set of instance variables. An object's state is created by the values assigned to these instance variables.

### Basic Syntax:

About Java programs, it is very important to keep in mind the following points.

- **Case Sensitivity:** Java is case sensitive, which means identifier **Hello** and **hello** would have different meaning in Java.

- **Class Names:** For all class names the first letter should be in Upper Case. If several words are used to form a name of the class, each inner word's first letter should be in Upper Case.

  **Example**

  *class MyFirstJavaClass*

- **Method Names:** All method names should start with a Lower Case letter.If several words are used to form the name of the method, then each inner word's first letter should be in Upper Case.

**Example**

*public void myMethodName()*

- **Program File Name -** Name of the program file should exactly match the class name.

## Java Identifiers:

All Java components require names. Names used for classes, variables and methods are called identifiers.

In Java, there are several points to remember about identifiers. They are as follows:

- All identifiers should begin with a letter (A to Z or a to z), currency character ($) or an underscore (_).
- After the first character, identifiers can have any combination of characters.
- A key word cannot be used as an identifier.
- Most importantly, identifiers are case sensitive.

  **Examples of legal identifiers**: age, $salary, _value, __1_value

- **Examples of illegal identifiers**: 123abc, -salary

## Java Modifiers:

Like other languages, it is possible to modify classes, methods, etc., by using modifiers. There are two categories of modifiers:

- **Access Modifiers:** default, public , protected, private
- **Non-access Modifiers:** final, abstract, strictfp

## Java Keywords:

The following list shows the reserved words in Java. These reserved words may not be used as constant or variable or any other identifier names.

| | | | |
|---|---|---|---|
| abstract | assert | boolean | break |
| byte | case | catch | char |
| class | const | continue | default |
| do | double | else | enum |
| extends | final | finally | float |
| for | goto | if | implements |
| import | instanceof | int | interface |
| long | native | new | package |
| private | protected | public | return |
| short | static | strictfp | super |
| switch | synchronized | this | throw |
| throws | transient | try | void |
| volatile | while | | |

## Java - Basic Data types

There are two data types available in Java:

- **Primitive Data Types**
- **Reference/Object Data Types**

### Primitive Data Types:

There are eight primitive data types supported by Java. Primitive data types are predefined by the language and named by a keyword.

byte: **byte data type is an 8-bit signed two's complement integer.**

Example: byte a = 100 , byte b = -50

short: **short data type is a 16-bit signed two's complement integer.**

- Example: short s = 10000, short r = -20000

int: **int data type is a 32-bit signed two's complement integer.**

- Example: int a = 100000, int b = -200000

long: **long data type is a 64-bit signed two's complement integer.**

- Example: long a = 100000L, long b = -200000L

float: **float data type is a single-precision 32-bit IEEE 754 floating point.**

- Example: float f1 = 234.5f

double: **double data type is a double-precision 64-bit IEEE 754 floating point.**

- Example: double d1 = 123.4

boolean: **boolean data type represents one bit of information.**

- Example: boolean one = true

char: **char data type is a single 16-bit Unicode character.**

- Example: char letterA ='A'

**Reference Data Types:**

- Reference variables are created using defined constructors of the classes. They are used to access objects..
- Class objects, and various type of array variables come under reference data type.
- Example: Animal animal = new Animal("giraffe");

## Final Variables

The value of a final variable cannot be change after it has been initialized. Such variables are similar to constants in other programming languages.

To declare a final variable, use the final keyword in the variable declaration before the type:

**final** int aFinalVar = 0;

The previous statement declares **aFinal** variable and initializes it, all at once. Subsequent attempts to assign a value to **aFinalVar** result in a compiler error. You may, if necessary, defer initialization of a final local variable. Simply declare the local variable and initialize it later, like this:

**final** int blankfinal;

…

blankfinal = 0;

A final local variable that has been declared but not yet initialized is called a **blankfinal**. Again, once a final local variable has been initialized, it cannot be set, and any later attempts to assign a value to **blankfinal** result in a compile-time error.

## Java - Basic Operators

### The Arithmetic Operators:

Assume integer variable A holds 10 and variable B holds 20, then:

| Operator | Description | Example |
|---|---|---|
| + | Addition - Adds values on either side of the operator | A + B will give 30 |
| - | Subtraction - Subtracts right hand operand from left hand operand | A - B will give -10 |
| * | Multiplication - Multiplies values on either side of the operator | A * B will give 200 |
| / | Division - Divides left hand operand by right hand operand | B / A will give 2 |
| % | Modulus - Divides left hand operand by right hand operand and returns remainder | B % A will give 0 |
| ++ | Increment - Increases the value of operand by 1 | B++ gives 21 |
| -- | Decrement - Decreases the value of operand by 1 | B-- gives 19 |

The following simple example program demonstrates the arithmetic operators.

```
public class Test {
  public static void main(String args[]) {
     int a = 10;
     int b = 20;
     int c = 25;
     int d = 25;
     System.out.println("a + b = " + (a + b) );
     System.out.println("a - b = " + (a - b) );
     System.out.println("a * b = " + (a * b) );
     System.out.println("b / a = " + (b / a) );
     System.out.println("b % a = " + (b % a) );
     System.out.println("c % a = " + (c % a) );
     System.out.println("a++   = " +  (a++) );
     System.out.println("b--   = " +  (a--) );
     // Check the difference in d++ and ++d
     System.out.println("d++   = " +  (d++) );
     System.out.println("++d   = " +  (++d) );     }  }
```

This would produce the following result:

**a + b = 30**

**a - b = -10**

**a * b = 200**

**b / a = 2**

**b % a = 0**

**c % a = 5**

**a++ = 10**

**b-- = 11**

**d++ = 25**

**++d = 27**

**The Relational Operators:**

| Operator | Description | Example |
|---|---|---|
| == | Checks if the values of two operands are equal or not, if yes then condition becomes true. | (A == B) is not true. |
| != | Checks if the values of two operands are equal or not, if values are not equal then condition becomes true. | (A != B) is true. |
| > | Checks if the value of left operand is greater than the value of right operand, if yes then condition becomes true. | (A > B) is not true. |
| < | Checks if the value of left operand is less than the value of right operand, if yes then condition becomes true. | (A < B) is true. |
| >= | Checks if the value of left operand is greater than or equal to the value of right operand, if yes then condition becomes true. | (A >= B) is not true. |
| <= | Checks if the value of left operand is less than or equal to the value of right operand, if yes then condition becomes true. | (A <= B) is true. |

The following simple example program demonstrates the relational operators.

```
public class Test {
  public static void main(String args[]) {
    int a = 10;
    int b = 20;
    System.out.println("a == b = " + (a == b) );
    System.out.println("a != b = " + (a != b) );
    System.out.println("a > b = " + (a > b) );
    System.out.println("a < b = " + (a < b) );
```

```
    System.out.println("b >= a = " + (b >= a) );
    System.out.println("b <= a = " + (b <= a) );
  }
}
```

This would produce the following result:

**a == b = false**

**a != b = true**

**a > b = false**

**a < b = true**

**b >= a = true**

**b <= a = false**

**The Bitwise Operators:**

Java defines several bitwise operators, which can be applied to the integer types, long, int, short, char, and byte. bitwise operator works on bits and performs bit-by-bit operation.

Assume if a = 60; and b = 13; now in binary format they will be as follows:

a = 0011 1100

b = 0000 1101

| Operator | Description | Example |
|---|---|---|
| & | Binary AND Operator copies a bit to the result if it exists in both operands. | (A & B) will give 12 which is 0000 1100 |
| \| | Binary OR Operator copies a bit if it exists in either operand. | (A \| B) will give 61 which is 0011 1101 |
| ^ | Binary XOR Operator copies the bit if it is set in one operand but not both. | (A ^ B) will give 49 which is 0011 0001 |
| ~ | Binary Ones Complement Operator is unary and has the effect of 'flipping' bits. | (~A ) will give -61 which is 1100 0011 in 2's complement form due to a signed binary number. |
| << | Binary Left Shift Operator. The left operands value is moved left by the number of bits specified by the right operand. | A << 2 will give 240 which is 1111 0000 |
| >> | Binary Right Shift Operator. The left operands value is moved right by the number of bits specified by the right operand. | A >> 2 will give 15 which is 1111 |
| >>> | Shift right zero fill operator. The left operands | A >>>2 will give 15 which is |

| | | |
|---|---|---|
| | value is moved right by the number of bits specified by the right operand and shifted values are filled up with zeros. | 0000 1111 |

The following simple example program demonstrates the bitwise operators:

```
public class Test {
  public static void main(String args[]) {
    int a = 60;         /* 60 = 0011 1100 */
    int b = 13;         /* 13 = 0000 1101 */
    int c = 0;
    c = a & b;       /* 12 = 0000 1100 */

    System.out.println("a & b = " + c );
    c = a | b;       /* 61 = 0011 1101 */
    System.out.println("a | b = " + c );
    c = a ^ b;       /* 49 = 0011 0001 */
    System.out.println("a ^ b = " + c );
    c = ~a;          /*-61 = 1100 0011 */
    System.out.println("~a = " + c );
    c = a << 2;     /* 240 = 1111 0000 */
    System.out.println("a << 2 = " + c );
    c = a >> 2;     /* 215 = 1111 */
    System.out.println("a >> 2  = " + c );
    c = a >>> 2;     /* 215 = 0000 1111 */
    System.out.println("a >>> 2 = " + c );    }}
```

This would produce the following result:

```
a & b = 12
a | b = 61
a ^ b = 49
~a = -61
a << 2 = 240
a >> 15
a >>> 15
```

**The Logical Operators:**

The following table lists the logical operators:

Assume Boolean variables A holds true and variable B holds false, then:

| Operator | Description | Example |
|---|---|---|
| && | Called Logical AND operator. If both the operands are non-zero, then the condition becomes true. | (A && B) is false. |
| \|\| | Called Logical OR Operator. If any of the two operands are non-zero, then the condition becomes true. | (A \|\| B) is true. |
| ! | Called Logical NOT Operator. Use to reverses the logical state of its operand. If a condition is true then Logical NOT operator will make false. | !(A && B) is true. |

The following simple example program demonstrates the logical operators.

```
public class Test {
  public static void main(String args[]) {
     boolean a = true;
     boolean b = false;
     System.out.println("a && b = " + (a&&b));
     System.out.println("a || b = " + (a||b) );
     System.out.println("!(a && b) = " + !(a && b));
  } }
```

This would produce the following result:

**a && b = false**

**a || b = true**

**!(a && b) = true**

**The Assignment Operators:**

There are following assignment operators supported by Java language:

| Operator | Description | Example |
|---|---|---|
| = | Simple assignment operator, Assigns values from right side operands to left side operand | C = A + B will assign value of A + B into C |

| | | |
|---|---|---|
| += | Add AND assignment operator, It adds right operand to the left operand and assign the result to left operand | C += A is equivalent to C = C + A |
| -= | Subtract AND assignment operator, It subtracts right operand from the left operand and assign the result to left operand | C -= A is equivalent to C = C - A |
| *= | Multiply AND assignment operator, It multiplies right operand with the left operand and assign the result to left operand | C *= A is equivalent to C = C * A |
| /= | Divide AND assignment operator, It divides left operand with the right operand and assign the result to left operand | C /= A is equivalent to C = C / A |
| %= | Modulus AND assignment operator, It takes modulus using two operands and assign the result to left operand | C %= A is equivalent to C = C % A |
| <<= | Left shift AND assignment operator | C <<= 2 is same as C = C << 2 |
| >>= | Right shift AND assignment operator | C >>= 2 is same as C = C >> 2 |
| &= | Bitwise AND assignment operator | C &= 2 is same as C = C & 2 |
| ^= | bitwise exclusive OR and assignment operator | C ^= 2 is same as C = C ^ 2 |
| \|= | bitwise inclusive OR and assignment operator | C \|= 2 is same as C = C \| 2 |

The following simple example program demonstrates the assignment operators.

```
public class Test {
  public static void main(String args[]) {
    int a = 10;
```

```
    int b = 20;
    int c = 0;
    c = a + b;
    System.out.println("c = a + b = " + c );
    c += a ;
    System.out.println("c += a  = " + c );
    c -= a ;
    System.out.println("c -= a = " + c );
    c *= a ;
    System.out.println("c *= a = " + c );
    a = 10;
    c = 15;
    c /= a ;
    System.out.println("c /= a = " + c );
    a = 10;
    c = 15;
    c %= a ;
    System.out.println("c %= a  = " + c );
    c <<= 2 ;
    System.out.println("c <<= 2 = " + c );
    c >>= 2 ;
    System.out.println("c >>= 2 = " + c );
    c >>= 2 ;
    System.out.println("c >>= a = " + c );
    c &= a ;
    System.out.println("c &= 2  = " + c );
    c ^= a ;
    System.out.println("c ^= a   = " + c );
    c |= a ;
    System.out.println("c |= a   = " + c );  }
}
```

This would produce the following result:

**c = a + b = 30**

**c += a  = 40**

**c -= a = 30**

**c *= a = 300**

**c /= a = 1**

**c %= a = 5**

**c <<= 2 = 20**

**c >>= 2 = 5**

**c >>= 2 = 1**

**c &= a = 0**

**c ^= a = 10**

**c |= a = 10**

**Miscellaneous Operators**

There are few other operators supported by Java Language.

Conditional Operator ( ? : )

Conditional operator is also known as the ternary operator. This operator consists of three operands and is used to evaluate Boolean expressions. The goal of the operator is to decide which value should be assigned to the variable. The operator is written as:

```
variable x = (expression) ? value if true : value if false
```

Following is the example:

```
public class Test {
  public static void main(String args[]){
    int a, b;
    a = 10;
    b = (a == 1) ? 20: 30;
    System.out.println( "Value of b is : " +  b );

    b = (a == 10) ? 20: 30;
    System.out.println( "Value of b is : " + b );
  }
}
```

This would produce the following result:

**Value of b is : 30**

**Value of b is : 20**

**instance of Operator:**

This operator is used only for object reference variables. The operator checks whether the object is of a particular type (class type or interface type). instanceof operator is written as:

```
( Object reference variable ) instanceof  (class/interface type)
```

If the object referred by the variable on the left side of the operator passes the IS-A check for the class/interface type on the right side, then the result will be true. Following is the example:

```
public class Test {
  public static void main(String args[]){
    String name = "James";
    // following will return true since name is type of String
    boolean result = name instanceof String;
    System.out.println( result );
  }
}
```

This operator will still return true if the object being compared is the assignment compatible with the type on the right.

## Precedence of Java Operators

Operator precedence determines the grouping of terms in an expression. This affects how an expression is evaluated. Certain operators have higher precedence than others; for example, the multiplication operator has higher precedence than the addition operator:

For example, x = 7 + 3 * 2; here x is assigned 13, not 20 because operator * has higher precedence than +, so it first gets multiplied with 3*2 and then adds into 7.

Here, operators with the highest precedence appear at the top of the table, those with the lowest appear at the bottom. Within an expression, higher precedence operators will be evaluated first.

| Category | Operator | Associativity |
|---|---|---|
| Postfix | () [] . (dot operator) | Left to right |
| Unary | ++ - - ! ~ | Right to left |
| Multiplicative | * / % | Left to right |
| Additive | + - | Left to right |
| Shift | >> >>> << | Left to right |
| Relational | > >= < <= | Left to right |
| Equality | == != | Left to right |
| Bitwise AND | & | Left to right |
| Bitwise XOR | ^ | Left to right |
| Bitwise OR | \| | Left to right |
| Logical AND | && | Left to right |
| Logical OR | \|\| | Left to right |
| Conditional | ?: | Right to left |
| Assignment | = += -= *= /= %= >>= <<= &= ^= \|= | Right to left |
| Comma | , | Left to right |

## Character Escape Codes

Following table shows the Java escape sequences:

| Escape Sequence | Description |
|---|---|
| \t | Inserts a tab in the text at this point. |
| \b | Inserts a backspace in the text at this point. |
| \n | Inserts a newline in the text at this point. |
| \r | Inserts a carriage return in the text at this point. |
| \f | Inserts a form feed in the text at this point. |
| \' | Inserts a single quote character in the text at this point. |
| \" | Inserts a double quote character in the text at this point. |

| | |
|---|---|
| \\ | Inserts a backslash character in the text at this point. |

When an escape sequence is encountered in a print statement, the compiler interprets it accordingly.

**Example:**

If you want to put quotes within quotes you must use the escape sequence, \", on the interior quotes:

```
public class Test {
  public static void main(String args[]) {
    System.out.println("She said \"Hello!\" to me.");
  } }
```

This would produce the following result: **She said "Hello!" to me.**

**Example**

```
public class example {
        public static void main(String[ ] args) {

        System.out.println("First line \n Second line");
      System.out.println("A \t B \t C ");
      System.out.println("D \t E \t F ");    } }
```

**Predictive Output:**

**First line**

**Second line**

**A B C**

**D E F**

## Classes in Java:

A class is a blue print from which individual objects are created.

A sample of a class is given below:

```
public class Dog{
  String breed;
  int ageC
  String color;
  void barking(){
  }
    void hungry(){
  }
    void sleeping(){
  }
}
```

A class can contain any of the following variable types.

- **Local variables:** Variables defined inside methods, constructors or blocks are called local variables. The variable will be declared and initialized within the method and the variable will be destroyed when the method has completed.

```
public class Test{
  public void pupAge(){
    int age = 0;
    age = age + 7;
    System.out.println("Puppy age is : " + age);
  }
    public static void main(String args[]){
    Test test = new Test();
    test.pupAge();
  }
}
```

- **Instance variables:** Instance variables are variables within a class but outside any method. These variables are initialized when the class is instantiated. Instance variables can be accessed from inside any method, constructor or blocks of that particular class.

```
public class Employee{
  // this instance variable is visible for any child class.
  public String name;
    // salary  variable is visible in Employee class only.
  private double salary;
    // The name variable is assigned in the constructor.
  public Employee (String empName){
    name = empName;
  }
  // The salary variable is assigned a value.
  public void setSalary(double empSal){
    salary = empSal;
  }
    // This method prints the employee details.
  public void printEmp(){
    System.out.println("name  : " + name );
    System.out.println("salary :" + salary);
  }
  public static void main(String args[]){
    Employee empOne = new Employee("Ransika");
    empOne.setSalary(1000);
    empOne.printEmp();
  }
}
```

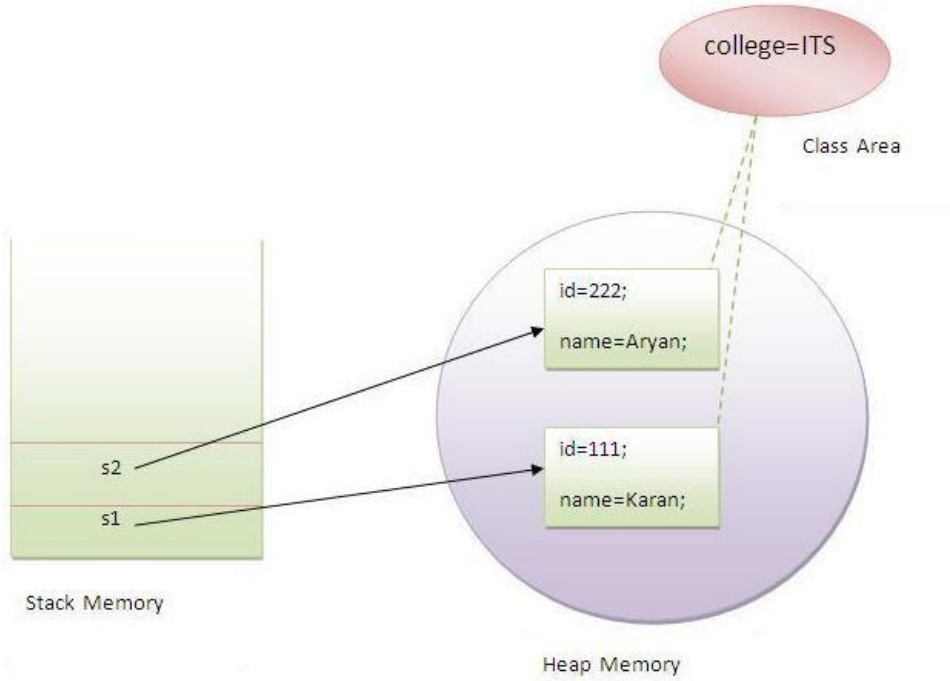
- **Class variables:** Class variables are variables declared with in a class, outside any method, with the static keyword.

```
public class Employee{
  // salary  variable is a private static variable
  private static double salary;
  // DEPARTMENT is a constant
  public static final String DEPARTMENT = "Development ";
  public static void main(String args[]){
    salary = 1000;
    System.out.println(DEPARTMENT + "average salary:" + salary);
  }
}
```

**static variable**

**Example 1**

```
class Student {
      int rollno;
      String name;
      static String college = "ITS";

      Student(int r,String n){
         rollno = r;
         name = n;
         }

      void display() {
            System.out.println(rollno + " " + name + " " +
college);
      }

      public static void main(String args[]) {
            Student s1 = new Student(111, "Karan");
            Student s2 = new Student(222, "Aryan");
            s1.display();
            s2.display();
      }
}
```




**Example 2**

```
class Counter {
	static int count = 0;
	Counter() {
		count++;
		System.out.println(count);
	}

	public static void main(String args[]) {

		Counter c1 = new Counter();
		Counter c2 = new Counter();
		Counter c3 = new Counter();

	}
}
```

## Declaring Array Variables:

```
dataType[] arrayRefVar;   // preferred way.
or
dataType arrayRefVar[];  //  works but not preferred way.
dataType[] arrayRefVar = new dataType[arraySize];
```

```
public class TestArray {
  public static void main(String[] args) {
    double[] myList = {1.9, 2.9, 3.4, 3.5};
    // Print all the array elements
    for (int i = 0; i < myList.length; i++) {
      System.out.println(myList[i] + " ");
    }
    // Summing all elements
    double total = 0;
    for (int i = 0; i < myList.length; i++) {
      total += myList[i];
    }
    System.out.println("Total is " + total);
    // Finding the largest element
    double max = myList[0];
    for (int i = 1; i < myList.length; i++) {
      if (myList[i] > max) max = myList[i];
    }
    System.out.println("Max is " + max);
  }
}
```

## Creating Strings:

The most direct way to create a string is to write:

```
String greeting = "Hello world!";
```

```
public class StringDemo{

  public static void main(String args[]){

    char[] helloArray = { 'h', 'e', 'l', 'l', 'o', '.'};

    String helloString = new String(helloArray);

    System.out.println( helloString );

  }

}
```

```
  public class StringDemo {
public static void main(String args[]) {
    String palindrome = "Dot saw I was Tod";
    int len = palindrome.length();
    System.out.println( "String Length is : " + len );
  }
}
```

## Constructors:

Every class has a constructor. If we do not explicitly write a constructor for a class the Java compiler builds a default constructor for that class.

Each time a new object is created, at least one constructor will be invoked. The main rule of constructors is that they should have the same name as the class. A class can have more than one constructor.

Example of a constructor is given below:

```
public class Puppy{
  public Puppy(){
  }
  public Puppy(String name){
    // This constructor has one parameter, name.
  }
}
```

## Creating an Object:

As mentioned previously, a class provides the blueprints for objects. So basically an object is created from a class. In Java, the new key word is used to create new objects.

There are three steps when creating an object from a class:

- **Declaration:** A variable declaration with a variable name with an object type.
- **Instantiation:** The 'new' key word is used to create the object.
- **Initialization:** The 'new' keyword is followed by a call to a constructor. This call initializes the new object.

**Example of creating an object is given below:**

```
public class Puppy{
  public Puppy(String name){
     // This constructor has one parameter, name.
     System.out.println("Passed Name is :" + name );
  }
    public static void main(String []args){
     // Following statement would create an object myPuppy
     Puppy myPuppy = new Puppy( "tommy" );
  }
}
```

## Accessing Instance Variables and Methods:

Instance variables and methods are accessed via created objects. To access an instance variable the fully qualified path should be as follows:

```
/* First create an object */
ObjectReference = new Constructor();

/* Now call a variable as follows */
ObjectReference.variableName;

/* Now you can call a class method as follows */
ObjectReference.MethodName();
```

### Example:

This example explains how to access instance variables and methods of a class:

```
public class Puppy{
     int puppyAge;
  public Puppy(String name){
     // This constructor has one parameter, name.
     System.out.println("Name chosen is :" + name );
  }
     public void setAge( int age ){
      puppyAge = age;
```

```
   }
   public int getAge( ){
      System.out.println("Puppy's age is :" + puppyAge );
      return puppyAge;
   }
     public static void main(String []args){
     /* Object creation */
     Puppy myPuppy = new Puppy( "tommy" );

     /* Call class method to set puppy's age */
     myPuppy.setAge( 2 );

     /* Call another class method to get puppy's age */
     myPuppy.getAge( );

     /* You can access instance variable as follows as well */
     System.out.println("Variable Value :" + myPuppy.puppyAge );
   }
}
```

**Flow control**

- There are three types of **loop statements**: iteration statements (**for**, **while**, and **do-while**) create loops
- selection statements (**switch** and all the **if** statements) tell the program under what circumstances the program will use statements
- jump statements (**break**, **continue**, and **return**) shift control out to another part of the program.

**Loops**

- **The while loop**

while ( <boolean condition statement> ) {
<code to execute as long as that condition is true>
}

**Example:**

```
public class TestWhile {
	public static void main(String[] arg) {
		int x = 0; // Initiates x at 0.
		while (x < 10) { // Boolean condition statement.
			System.out.println("Looping");
			x++; // Increments x for the next iteration.
		}
	}
}
```

**The do-while loop**

**Example:**

```
public class TestDoWhile {
	public static void main(String[] arg) {
		int x = 0;
		do {
			System.out.println("Looping");
			x++;
		} while (x < 10);
	}
}
```

**The for loop**

for ( <initialization> ; <boolean condition> ; <iteration> ) {
<execution code>
}

**Example:**

```
public class TestFor {
	public static void main(String[] arg) {
		for (int x = 0; x < 10; x++) {
			System.out.println("Looping");
		}
}	}
```

**Loop control statements**

These statements add control to the loop statements.

— **The break statement:** The break statement will allow you to exit a loop structure before the test condition is met. Once a break statement is encountered, the loop immediately terminates, skipping any remaining code. For instance:

**Example:**

```
public class TestBreak {
	public static void main(String[] arg) {
		int x = 0;
		while (x < 10) {
			System.out.println("Looping");
			x++;
			if (x == 5)
				break;
			else
				System.out.println(x);
		}
	}

}
```

— **The continue statement:** The continue statement is used to skip the rest of the loop and resume execution at the next loop iteration.

**Example:**

```
public class TestContinue {
	public static void main(String[] arg) {
		for (int x = 0; x < 10; x++) {
			if (x == 5)
			    continue;//go back to beginning of loop
with x=6
			System.out.println(x + " Looping");
		}

	}

}
```

## Enhanced for-loop (or "for-each" Loop)

### Conditional statements

Conditional statements are used to provide your code with decision making capabilities. There are two conditional structures in Java: the **if-else** statement, and the **switch** statement.

— **The if-else statement**

The syntax of an if-else statement is as follows:

if (<condition1>) {
... //code block 1
}
else if (<condition2>) {
... //code block 2
}
else {
... //code block 3}

**Example**

```
public class TestIf {
      public static void main(String[] arg) {
```

| Syntax | Example |
|---|---|
| `for ( type item : anArray ) {`<br>`  body ;`<br>`}`<br>`// type must be the same as the`<br>`// anArray's type` | `int[] numbers = {8, 2, 6, 4, 3};`<br>`int sum = 0;`<br>`// for each int number in int[] numbers`<br>`for (int number : numbers) {`<br><br>`  sum += number;`<br>`}`<br>`System.out.println("The sum is " + sum);` |

```
            int x = 5, y = 6;
            if (x % 2 == 0)
                  System.out.println("x is even");
            else
                  System.out.println("x is odd");
            if (x == y)
                  System.out.println("x equals y");
            else if (x < y)
                  System.out.println("x is less than y");
            else
                  System.out.println("x is greater than y");

      }

}
```

— **The switch statement**

switch (<expression>){
case <value1>: <codeBlock1>;
break;
case <value2>: <codeBlock2>;
break;
default : <codeBlock3>;
}

**Example**:

```
public class TestSwitch {
	public static void main(String[] arg) {
		char c='k';
		switch (c) {
		case '1':
		case '3':
		case '5':
		case '7':
		case '9':
			System.out.println("c is an odd number");
			break;
		case '0':
		case '2':
		case '4':
		case '6':
		case '8':
			System.out.println("c is an even number");
	break;
		case ' ':
			System.out.println("c is a space");
			break;
		default:
			System.out.println("c is not a number or a
space");
		}

	}
}
```

**Terminating Program**

**System.exit(int exitCode)**

**String**

A String is a sequence of characters.

```
String s1 = "Hi, This is a string!"  // String literals are enclosed in double quotes
String s2 = ""                       // An empty string
```

**String and '+' Operator**

In Java, '+' is a special operator. It is *overloaded*. *Overloading* means that it carries out different operations depending on the types of its two operands.

- If both operands are numbers (byte, short, int, long, float, double, char), '+' performs the usual *addition*, e.g.,

```
• 1 + 2 → 3
• 1.2 + 2.2 → 3.4
```

```
1 + 2.2 → 1.0 + 2.2 → 3.2
```

- If both operands are `Strings`, `'+'` *concatenates* the two Strings and returns the concatenated `String`. E.g.,

```
• "Hello" + "world" → "Helloworld"
```

```
"Hi" + ", " + "world" + "!" → "Hi, world!"
```

- If one of the operand is a String and the other is numeric, the numeric operand will be converted to String and the two Strings concatenated, e.g.,

```
• "The number is " + 5 → "The number is " + "5" → "The number is 5"
• (suppose average=5.5)
 "The average is " + average + "!"  → "The average is 5.5!"
```

```
(suppose a=1, b=1)
"How about " + a + b → "How about 11"
```

**String Operations**

The most commonly-used String methods are:

- length(): return the length of the string.
- charAt(int index): return the char at the index position (index begins at 0 to length()-1).
- equals(): for comparing the contents of two strings. Take note that you cannot use "==" to compare two strings.

For examples,

```
String str = "Java is cool!";
System.out.println(str.length());       // return int 13
System.out.println(str.charAt(2));      // return char 'v'
System.out.println(str.charAt(5));      // return char 'i'
```

```
// Comparing two Strings
String anotherStr = "Java is COOL!";
System.out.println(str.equals(anotherStr));                    //
return boolean false
System.out.println(str.equalsIgnoreCase(anotherStr));          //
return boolean true
System.out.println(anotherStr.equals(str));                    //
return boolean false
System.out.println(anotherStr.equalsIgnoreCase(str));          //
return boolean true
// (str == anotherStr) to compare two Strings is WRONG!!!
```

### String/Primitive Conversion

**"String" to "int/byte/short/long"**:

You could use the JDK built-in methods **Integer.parseInt(*anIntStr*)** to convert a String containing a valid integer literal (e.g., "1234") into an int (e.g., 1234). For example,

```
String inStr = "5566";
int number = Integer.parseInt(inStr);   // number <- 5566
```

**"String" to "double/float"**: You could use **Double.parseDouble(*aDoubleStr*)** or Float.parseFloat(*aFloatStr*) to convert a String (containing a floating-point literal) into a double or float, e.g.

```
String inStr = "55.66";
float aFloat = Float.parseFloat(inStr);           // aFloat <-
55.66f
double aDouble = Double.parseDouble("1.2345");   // aDouble <-
1.2345
aDouble = Double.parseDouble("abc");                // Runtime
Error: NumberFormatException
```

**"String" to "char"**: You can use *aStr*.charAt(*index*) to extract individual character from a String, e.g.,

```
// Converting from binary to decimal
String msg = "101100111001!";
int pos = 0;
while (pos < msg.length()) {
   char binChar = msg.charAt(pos);    // Extract character at
pos
   // Do something about the character
   .......
   ++pos;
```

```
}
```

**"String" to "boolean"**: You can use
method **Boolean.parseBoolean(*aBooleanStr*)** to convert string of "true" or "false" to boolean true or false, e.g.,

```
String boolStr = "true";
boolean done = Boolean.parseBoolean(boolStr);  // done <- true
boolean valid = Boolean.parseBoolean("false"); // valid <- false
```

**Primitive (int/double/float/byte/short/long/char/boolean) to "String"**: To convert a primitive to a String,
you can use the '+' operator to concatenate the primitive with an *empty*String (""),
or
use the JDK built-in methods
**String.valueOf(*aPrimitve*)**
Integer.toString(*anInt*),
Double.toString(*aDouble*),
Character.toString(*aChar*),
Boolean.toString(*aBoolean*), etc.
For example,

```
// String concatenation operator '+' (applicable to ALL primitive types)
String s1 = 123 + "";    // int 123 -> "123"
String s2 = 12.34 + "";  // double 12.34 -> "12.34"
String s3 = 'c' + "";    // char 'c' -> "c"
String s4 = true + "";   // boolean true -> "true"

// String.valueOf(aPrimitive) is applicable to ALL primitive types
String s1 = String.valueOf(12345);   // int 12345 -> "12345"
String s2 = String.valueof(true);    // boolean true -> "true"
double d = 55.66;
String s3 = String.valueOf(d);           // double 55.66 -> "55.66"

// toString() for each primitive type
String s4 = Integer.toString(1234);  // int 1234 -> "1234"
String s5 = Double.toString(1.23);   // double 1.23 -> "1.23"
char c1   = Character.toString('z'); // char 'z' -> "z"

// char to String
char c = 'a';
```

```
String s5 = c;       // Compilation Error: incompatible types
String s6 = c + "";  // Convert the char to String

// boolean to String
boolean done = false;
String s7 = done + "";    // boolean false -> "false"
String s8 = Boolean.toString(done);
String s9 = String.valueOf(done);
```

**Java Scanner class**

The Scanner class is a class in java.util, which allows the user to read values of various types.

**Numeric and String Methods**

| *Method* | *Returns* |
|---|---|
| int nextInt() | Returns the next token as an int. If the next token is not an integer,InputMismatchException is thrown. |
| long nextLong() | Returns the next token as a long. If the next token is not an integer,InputMismatchException is thrown. |
| float nextFloat() | Returns the next token as a float. If the next token is not a float or is out of range, InputMismatchException is thrown. |
| double nextDouble() | Returns the next token as a long. If the next token is not a float or is out of range, InputMismatchException is thrown. |
| String next() | Finds and returns the next complete token from this scanner and returns it as a string; a token is usually ended by whitespace such as a blank or line break. |
| String nextLine() | Returns the rest of the current line, excluding any line separator at the end. |
| void close() | Closes the scanner. |

**Boolean Methods**

| ***Method*** | ***Returns*** |
|---|---|
| boolean hasNextLine() | Returns true if the scanner has another line in its input; falseotherwise. |
| hasNext() | Returns true if this scanner has another token in its input. |
| boolean hasNextInt() | Returns true if the next token in the scanner can be interpreted as an int value. |
| boolean hasNextFloat() | Returns true if the next toke in the scanner can be interpreted as a float value. |

**Example 1:**

```
import java.util.Scanner;
class ScannerTest {
      public static void main(String args[]) {
            Scanner sc = new Scanner(System.in);
            System.out.println("Enter your rollno");
            int rollno = sc.nextInt();
            System.out.println("Enter your name");
```

```
            String name = sc.next();
            System.out.println("Enter your fee");
            double fee = sc.nextDouble();
            System.out.println("Rollno:" + rollno + " name:" +
name + " fee:" + fee);
            sc.close();
      }

}
```

**Example 2:**

```
import java.util.Scanner;
public class ScannerDemo {
   public static void main(String[] args) {
   String s = "Hello World! \n 3 + 3.0 = 6.0 true ";
   // create a new scanner with the specified String Object
   Scanner scanner = new Scanner(s);
   // print the next line
   System.out.println("" + scanner.nextLine());
   // print the next line again
   System.out.println("" + scanner.nextLine());
   // close the scanner
   scanner.close();
   }
}
```

## Exception Handling in Java

**Dictionary Meaning:** Exception is an abnormal condition.

**In java**, exception is an event that disrupts the normal flow of the program. It is an object, which is thrown at runtime.

Common scenarios where exceptions may occur

- **Scenario where ArithmeticException occurs:** If we divide any number by zero, there occurs an ArithmeticException.

  int a=50/0;

- **Scenario where NullPointerException occurs:** If we have null value in any variable, performing any operation by the variable occurs an NullPointerException.

  String s=null;

  System.out.println(s.length());//

- **Scenario where ArrayIndexOutOfBoundsException occurs:** If you are inserting any value in the wrong index, it would result ArrayIndexOutOfBoundsException as shown below:

  int a[]=new int[5];

  a[10]=50;

## Java Exception Handling Keywords

There are **five** keywords used in java exception handling: try, catch, finally, throw, throws

**Example:**

```
public class TestTryCatch {
	public static void main(String args[]) {
		try {
			int data = 50 / 0;
		} catch (Exception e) {
			System.out.println(e);
		}
		System.out.println("rest of the code...");
	}
}
```

Output:

java.lang.ArithmeticException: / by zero

rest of the code...

```
public class ExcepTest{


  public static void main(String args[]){
    try{
      int a[] = new int[2];
      System.out.println(a[3]);
    }catch(ArrayIndexOutOfBoundsException e){
      System.out.println("Exception:" + e);
    }
    System.out.println("Out of the block");
  }
}
```

**The Finally Block**

The finally block follows a try block or a catch block. A finally block of code always executes, irrespective of occurrence of an Exception.

Using a finally block allows you to run any cleanup-type statements that you want to execute, no matter what happens in the protected code.

A finally block appears at the end of the catch blocks and has the following syntax −

Syntax

```
try {
  // Protected code
}catch(ExceptionType1 e1) {
  // Catch block
}catch(ExceptionType2 e2) {
  // Catch block
}catch(ExceptionType3 e3) {
  // Catch block
}finally {
  // The finally block always executes.
}
```

```
public class ExcepTest {
  public static void main(String args[]) {
    int a[] = new int[2];
    try {
      System.out.println("Access element three :" + a[3]);
    }catch(ArrayIndexOutOfBoundsException e) {
      System.out.println("Exception thrown  :" + e);
    }finally {
      a[0] = 6;
      System.out.println("First element value: " + a[0]);
      System.out.println("The finally statement is executed");
    }
  }
}
```

## Output

```
Exception thrown  :java.lang.ArrayIndexOutOfBoundsException: 3
First element value: 6
The finally statement is executed
```

## this keyword in java

In java, this is a **reference variable** that refers to the current object.

If there is ambiguity between the instance variable and parameter, this keyword resolves the problem of ambiguity.

**Example:**

```
public class Student {
        int id;
        String name;
        Student(int id, String name) {
                id = id;
                name = name;
        }
        void display() {
                System.out.println(id + " " + name);
        }
        public static void main(String args[]) {
                Student s1 = new Student(111, "Karan");
                Student s2 = new Student(321, "Aryan");
                s1.display();
                s2.display();
        }
}
```

**Output: ?**

**Solution of the above problem by this keyword**

```
//example of this keyword
    public class Student {
            int id;
            String name;
            Student(int id, String name) {
                    this.id = id;
                    this.name = name;
            }
            void display() {
                    System.out.println(id + " " + name);
            }
            public static void main(String args[]) {
                    Student s1 = new Student(111, "Karan");
                    Student s2 = new Student(222, "Aryan");
                    s1.display();
                    s2.display();
            }
}
```

**Output: ?**

**this() can be used to invoked current class constructor.**

From within a constructor, you can also use the this keyword to call another constructor in the same class.

```
    public class Rectangle {
            private int x, y;
            private int width, height;
            public Rectangle() {
                    this(0, 0, 1, 1);
            }
            public Rectangle(int width, int height) {
                    this(0, 0, width, height);
            }
            public Rectangle(int x, int y, int width, int height) {
                    this.x = x;
                    this.y = y;
                    this.width = width;
                    this.height = height;
            }

            public static void main(String args[]) {
                    Rectangle Obj = new Rectangle(4, 6);
                    System.out.println("Variable values of object : ");
                    System.out.println("Object x =  " + Obj.x);
                    System.out.println("Object y =  " + Obj.y);
                    System.out.println("Object height =  " + Obj.height);
                    System.out.println("Object width = " + Obj.width);
            }
    }
    }
```

```
//Program of this() constructor call (constructor chaining)
class Student {
	int id;
	String name;
	Student() {
		System.out.println("default constructor is invoked");
	}
	Student(int id, String name) {
		this();// it is used to invoked current class constructor.
		this.id = id;
		this.name = name;
	}
	void display() {
		System.out.println(id + " " + name);
	}
	public static void main(String args[]) {
		Student e1 = new Student(111, "karan");
		Student e2 = new Student(222, "Aryan");
		e1.display();
		e2.display();
	}
}
```

Output:?

## Random Numbers

There are two principal means of generating random numbers:

- the Random class generates random integers, doubles, longs and so on, in various ranges.
- the static method **Math.**random generates *doubles* between 0 (inclusive) and 1 (exclusive).

## The Random class

A Random object generates pseudo-random numbers.

Class Random is found in the java.util package.

**nextInt()** returns a random integer

**nextInt(max)** returns a random integer in the range [0, max)

**nextDouble()** returns a random real number in the range [0.0, 1.0)

**Example**:

```
Random rand = new Random();
int randomNumber = rand.nextInt(10); //0-9
```

To get a random number from 1 to N

```
int n = rand.nextInt(20) + 1; // 1-20
```

To get a number in arbitrary range [min, max] inclusive:

nextInt(max - min +1) + min

**Example:** A random integer between 4 and 10 inclusive:

```
int n = rand.nextInt(7) + 4;
```

nextDouble method returns a double between 0.0 - 1.0

**Example:** Get a random number value between 1.5 and 4.0:

```
double randomGpa = rand.nextDouble() * 2.5 + 1.5;
```

Write a program that simulates rolling of two 6-sided dice until their combined result comes up as 7.

```
import java.util.Random;
public class Dice {
	public static void main(String[] args) {
		Random rand = new Random();
		int tries = 0;
		int sum = 0;
		while (sum != 7) { // roll the dice once
			int roll1 = rand.nextInt(6) + 1;
			int roll2 = rand.nextInt(6) + 1;
			sum = roll1 + roll2;
			System.out.println(roll1 + " + " + roll2 + " = " + sum);
			tries++;
		}
		System.out.println("You won after " + tries + " tries!");
	}
}
```

**Write a program that generates ten numbers between 0 to 99**

```
import java.util.Random;
public class RandomTest {
public static void main(String[] args){
	System.out.println("Generating 10 random integers in range 0..99.");
	Random randomGenerator = new Random();
	for (int idx = 1; idx <= 10; ++idx){
		int randomInt = randomGenerator.nextInt(100);
		System.out.println("Generated : " + randomInt);
		}
	System.out.println("Done.");
	}
}
```

**Use Math.random() to create random numbers**

**Example:**

```
public class RandomTest {
	public static  void main(String[] args){
	   for(int i =0; i<3; i++){
	     double randomDouble = Math.random();
```

```
            System.out.println("Random Number in Java: " + randomDouble);
          }
        }
 }
```

**Example:**

```
public class RandomInteger {
       public static  void main(String[] args){
         for(int i =0; i<3; i++){
            int randomInt = (int)(10.0 * Math.random());
            System.out.println("pseudo random int: " + randomInt );
        }
  }
 }
```

## Java AWT and Java Swing

Java GUI programming involves two packages: the original Abstract Windows Kit (AWT) and the newer Swing toolkit.

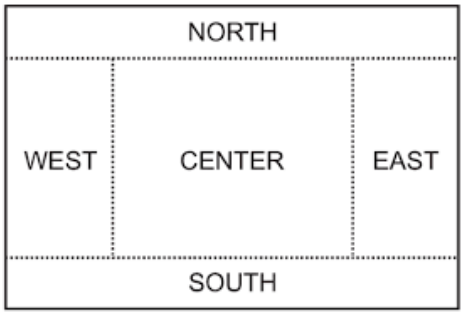
Java AWT is an API to develop GUI or window-based application in java. Java AWT components are platform-dependent and heavyweight.

Swing API is set of extensible GUI Components to ease developer's life to create Java based Front End/ GUI Applications. Java Swing is built on the top of AWT API. Unlike AWT, Java Swing provides platform-independent and lightweight components.

Every user interface considers the following three main aspects:

- **UI elements** : These are the core visual elements the user eventually sees and interacts with.
  - **JButton**:This class creates a labeled button.
  - **JLabel**: A JLabel object is a component for placing text in a container.
  - **JCheck Box**: A JCheckBox is a graphical component that can be in either an **on** (true) or **off** (false) state.
  - **JRadioButton**: The JRadioButton class is a graphical component that can be in either an **on** (true) or **off** (false) state. in a group.
  - **JList**: A JList component presents the user with a scrolling list of text items.
  - **JComboBox**: A JComboBox component presents the user with a to show up menu of choices.
  - **JTextField**: A JTextField object is a text component that allows for the editing of a single line of text.
  - **JPasswordField**: A JPasswordField object is a text component specialized for password entry.
  - **JTextArea**: A JTextArea object is a text component that allows for the editing of a multiple lines of text.
  - **JOptionPane**: JOptionPane provides set of standard dialog boxes that prompt users for a value or informs them of something.

- **Layouts:** They define how UI elements should be organized on the screen and provide a final look and feel to the GUI
  - **BorderLayout**: The borderlayout arranges the components to fit in the five regions: east, west, north, south and center.




— **FlowLayout**: It layouts the components in a directional flow.

— **GridLayout**: The GridLayout manages the components in form of a rectangular grid.

- **Behavior:** These are events, which occur when the user interacts with UI elements.

**JFrame**

A JFrame is an independent window that act as the main window of an application and contain other GUI components such as buttons and menus.

**Creating a JFrame Window**

1. Construct an object of the JFrame class.
2. Set the size of the Jframe.
3. Set the title of the Jframe to appear in the title bar (title bar will be blank if no title is set).
4. Make the Jframe visible.

**JFrame Constructor**

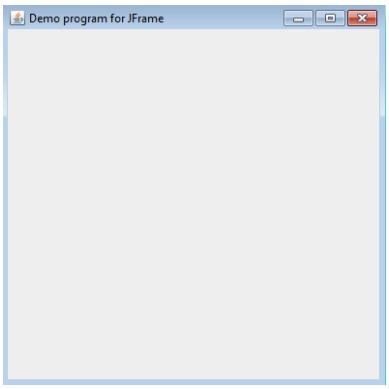
**JFrame():** Constructs a new frame that is initially invisible**.**

**JFrame(String title)**

JFrame(GraphicsConfiguration gc): Creates a Frame in the specified GraphicsConfiguration of a screen device and a blank title.

**Example**

```
import javax.swing.*;
public class SwingJFrameDemo {

        public static void main(String[] args) {
                JFrame frame = new JFrame("Demo program for JFrame");
                frame.setSize(400, 400);
                frame.setVisible(true);

        }
}
```




**Setting layout manager**

The default layout of the frame is **BorderLayout**, we can set another layout like this:

frame.setLayout(new GridLayout());

**Or**

frame.setLayout(new FlowLayout());

**the call** setLayout(layout) **is equivalent to this call:**

frame.getContentPane().setLayout(layout);

**Specifying window closing behavior**

We can specify which action will be executed when the user clicks on the frame's close button:

- **Do nothing (default):**

  frame.setDefaultCloseOperation(JFrame.DO_NOTHING_ON_CLOSE);

- **Hide the frame:**

  frame.setDefaultCloseOperation(JFrame.HIDE_ON_CLOSE);

- **Exit the program (JVM terminates):**

  frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

**Set background color**

frame.getContentPane().setBackground(Color.GREEN);

**JButton class:**

The JButton class is used to create a button that have platform-independent implementation with the following constructor methods:

**JButton()**: creates a button with no text and icon.

**JButton(String s)**:creates a button with the specified text.

**JButton(Icon i):creates a button with the specified icon object**

**Methods of Button class:**

**public void setText(String s):** is used to set specified text on button.

**public void setEnabled(boolean b):** is used to enable or disable the button.

**public void addActionListener(ActionListener a):** is used to add the action listener to this object.

**public String getText():** is used to return the text of the button.

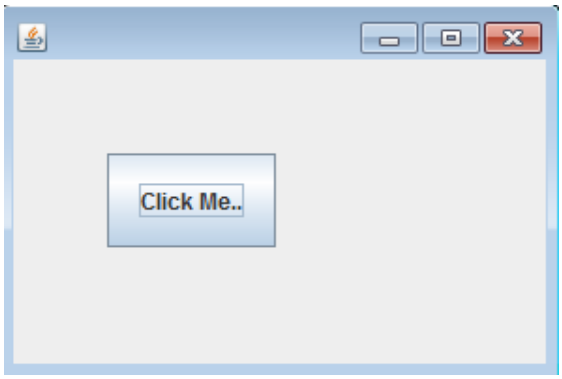
**public void setIcon(Icon b):** is used to set the specified Icon on the button.

**public Icon getIcon():**is used to get the Icon of the button.

**public void setMnemonic(int a):** is used to set the mnemonic on the button.

**Example**

```
import javax.swing.JButton;
import javax.swing.JFrame;

public class JButtonExample {
        JButtonExample() {

                JFrame frame = new JFrame();

                // Creating Button
                JButton but = new JButton("Click Me..");

                // This method specifies the location and size of button.
                but.setBounds(50, 50, 90, 50);

                // Adding button onto the frame
                frame.add(but);


                // Setting Frame size. This is the window size
                frame.setSize(300, 200);

                frame.setLayout(null);
                frame.setVisible(true);

                frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        }

        public static void main(String[] args) {
                JButtonExample jb = new JButtonExample();
        }
}
```




the call **add(Component)** is equivalent to this call:

frame.getContentPane().add(Component);

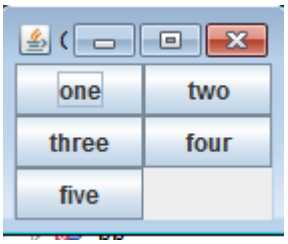
A **JButton** generally represents a button that, when clicked by the user, carries out a particular action. A **JButton** has a so-called **ActionListener** attached to it, which in effect defines the task to be performed when the button is clicked.

```
JButton bt = new JButton("Click Me..");
bt.addActionListener(new ActionListener() {
  public void actionPerformed(ActionEvent evt) {
    // ... called when button clicked
  }
});
```

**Example**

```
import java.awt.GridLayout;
import javax.swing.JButton;
import javax.swing.JFrame;

public class GridLayoutTest {

        public static void main(String[] args) {

                JFrame frame = new JFrame("GridLayout Test");
                frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
                frame.setLayout(new GridLayout(3, 2));
                JButton jb1 = new JButton("one");
                JButton jb2 = new JButton("two");
                JButton jb3 = new JButton("three");
                JButton jb4 = new JButton("four");
                JButton jb5 = new JButton("five");
                frame.getContentPane().add(jb1);
                frame.getContentPane().add(jb2);
                frame.getContentPane().add(jb3);
```

```
                frame.getContentPane().add(jb4);
                frame.getContentPane().add(jb5);
                frame.pack();
                frame.setVisible(true);
        }
}
```




**Example**

```
import java.awt.BorderLayout;
import javax.swing.JButton;
import javax.swing.JFrame;

public class BorderLayoutTest {

        BorderLayoutTest() {
                JFrame frame = new JFrame("BorderLayout Test");
                frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
                JButton jb1 = new JButton("one");
                JButton jb2 = new JButton("two");
                JButton jb3 = new JButton("three");
                JButton jb4 = new JButton("four");
                JButton jb5 = new JButton("five");
                frame.add(BorderLayout.NORTH,jb1);
                frame.add(BorderLayout. WEST,jb2);
                frame.add(BorderLayout. CENTER,jb3);
                frame.add(BorderLayout. EAST,jb4);
                frame.add(BorderLayout.SOUTH,jb5);
                frame.pack();
                frame.setVisible(true);
        }
        public static void main(String[] args) {
                BorderLayoutTest bt = new BorderLayoutTest();
        }
}
```

**JLabel Class**

The class **JLabel** can display either text, an image, or both.

**JLabel Constructor**

**JLabel()**: Creates a JLabel instance with no image and with an empty string for the title.

**JLabel(String text)**:Creates a JLabel instance with the specified text.

JLabel(Icon image):Creates a JLabel instance with the specified image.

## JPanel Class

**The JPanel is a virtual container that can be used to group certain components to create a user interface for Java Desktop Application.**

**Creating a JPanel:**

```
JPanel panel = new JPanel();
```

An example of how to set the layout manager when creating the panel.

```
JPanel panel = new JPanel(new BorderLayout());
```

**Or Setting layout manager**

Like other containers, a panel uses a layout manager to position and size its components. By default, a panel's layout manager is an instance of **FlowLayout**, which places the panel's contents in a row. You can easily make a panel use any other layout manager by invoking the setLayout method or by specifying a layout manager when creating the panel.

```
panel.setLayout(new BorderLayout());
```

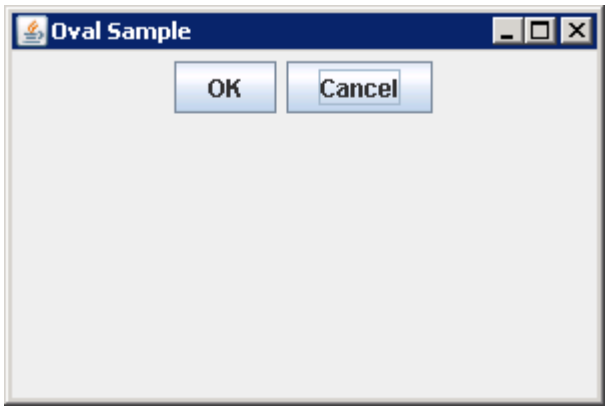
### Adding Components

```
panel.add(aComponent);
panel.add(anotherComponent);
```

When the layout manager is BorderLayout, you need to provide an argument specifying the added component's position within the panel. For example:

```
panel.add(aComponent, BorderLayout.CENTER);
panel.add(aComponent);
```

**Example**




```
import javax.swing.JButton;
import javax.swing.JFrame;
```

```
import javax.swing.JPanel;

        public class PanelWithComponents {
          public static void main(String[] a) {
            JPanel panel = new JPanel();
            JButton okButton = new JButton("OK");
            panel.add(okButton);
            JButton cancelButton = new JButton("Cancel");
            panel.add(cancelButton);
            JFrame frame = new JFrame("Oval Sample");
            frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
            frame.add(panel);
            frame.setSize(300, 200);
            frame.setVisible(true);
          }
        }
```

## JTextField Class

### Creating a JTextField object

When creating a text field component, it's common to specify some initial text and/or a number of columns from which the field's width is calculated.

**Create a text field with some initial text:**

```
JTextField textField = new JTextField("This is a text");
```

**Create a text field with a specified number of columns:**

```
JTextField textField = new JTextField(20);
```

**Create a text field with both initial text and number of columns:**

```
JTextField textField = new JTextField("This is a text", 20);
```

**Create a default and empty text field then set the text and the number of columns later:**

```
JTextField textField = new JTextField();
textField.setText("This is a text");
textField.setColumns(20);
```

**Adding the text field to a container**

A JTextField can be added to any container like JFrame or JPanel

```
frame.add(textField);
panel.add(textField);
```

**Add a JTextField to the container with a specific layout manager:**

```
frame.add(textField, BorderLayout.CENTER);
panel.add(textField, gridbagConstraints);
```

**Getting or setting content of the text field**

- Getting all content:

```
String content = textField.getText();
```

- Getting a portion of the content:

```
int offset = 5;
int length = 10;
 try {
   content = textField.getText(offset, length);
} catch (BadLocationException ex) {
   // invalid offset/length
     }
```

**Setting content:**

```
textField.setText("another text");
```

**Setting tooltip text**

Set tooltip for the text field as follows:

```
textField.setToolTipText("Please enter some text here");
```

**Adding event listeners**

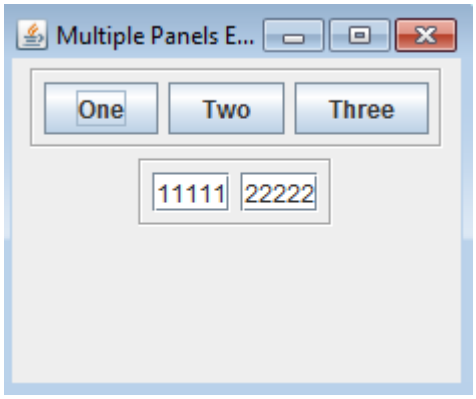
We can capture the event in which the user hits Enter key while typing in the text field. For example:

```
  textField.addActionListener(new ActionListener() {
   public void actionPerformed(ActionEvent event) {
      System.out.println("The entered text is: " + textField.getText());
   }
     });
textField.setForeground(Color.BLUE);
textField.setBackground(Color.YELLOW);
```

**Example:**

```
import javax.swing.JFrame; // A frame (main window)
import javax.swing.JPanel; // A container for other GUI components
import javax.swing.border.EtchedBorder; // A simple border
import java.awt.FlowLayout; // The simplest layout manager
```

```
import javax.swing.JButton; // A button
import javax.swing.JTextField; // A text field

public class MultiplePanels {
	public static void main(String[] args) {
		JFrame frame = new JFrame("Multiple Panels Example");
		frame.setSize(240, 200);
		frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
		frame.setLayout(new FlowLayout());

		// Create a JPanel object and add three buttons to it.
		JPanel panel1 = new JPanel();
		panel1.setBorder(new EtchedBorder()); // Give the panel a border
		panel1.add(new JButton("One"));
		panel1.add(new JButton("Two"));
		panel1.add(new JButton("Three"));

		// Create a JPanel object and add two text fields to it.
		JPanel panel2 = new JPanel();
		panel2.setBorder(new EtchedBorder()); // Give the panel a border
		panel2.add(new JTextField("11111"));
		panel2.add(new JTextField("22222"));

		// Add the two panels to this frame's content pane.
		frame.add(panel1);
		frame.add(panel2);

		frame.setVisible(true);

	}
}
```



## JPasswordField Class

JPasswordField is a subclass of the JTextField so it acts like an ordinary text field but it hides the actual characters typed by showing a series of echo characters such as asterisks (*) or a dots, for security purpose.

**Creating new JPasswordField object**

When creating a new JPasswordField object, we can specify an initial text and the field's width in number of columns, using one of the following constructors:

JPasswordField(int columns)

JPasswordField(String text)

JPasswordField(String text, int columns)

Note that the parameter columns specifies minimum width of the field in a number of columns, not in number of pixels nor characters. Here are some examples:

```
JPasswordField passwordField = new JPasswordField(20);
JPasswordField passwordField = new JPasswordField("secret");
JPasswordField passwordField = new JPasswordField("secret", 20);
```

We can also use an empty constructor to create new object and then set the columns and initial password later, for example:

```
JPasswordField passwordField = new JPasswordField(20);
passwordField.setColumns(10);
passwordField.setText("secret");
```

**Adding the password field to a container**

The password field cannot stand-alone. It must be added to a parent container such as JFrame or JPanel. For example:

```
frame.add(passwordField);
panel.add(passwordField);
```

**Setting and getting password**

Use the setText() method (inherited from javax.swing.text.JTextComponent class) to set password for the field:

```
passwordField.setText("secret");
```

To retrieve password typed into the field, use the getPassword() method:

```
    char[] password = passwordField.getPassword();
char[] password = passwordField.getPassword();
char[] correctPass = new char[] {'s', 'e', 'c', 'r', 'e', 't'};
 if (Arrays.equals(password, correctPass)) {
   System.out.println("Password is correct");
} else {
   System.out.println("Incorrect password");
}
```

## JOptionPane Class

JOptionPane Facilitates data entry and data output. class JOptionPane contains methods that display a dialog box.

**String showInputDialog**

- prompt for input
- return a string

**void showMessageDialog**

- display a message
- wait for an acknowledgement

int showConfirmDialog

- ask a yes/no question
- wait for a response

**JOptionPane**.showMessageDialog (null, "Put your message here");

to show text in different rows:

**JOptionPane**.showMessageDialog (**null**, "Put **n**your **n**message **n**here");

To show different messages with different icons, you'll need 4 arguments instead of 2.

**JOptionPane**.showMessageDialog (**null**, "Messagehere", "Titlehere", **JOptionPane**.PLAIN_MESSAGE);

There are different kinds of icons for a dialog box, just replace the last argument:

**JOptionPane**.PLAIN_MESSAGE *// this is a plain message*

**JOptionPane**.INFORMATION_MESSAGE *// this is a info message*

**JOptionPane**.ERROR_MESSAGE *// this is a error message*

**JOptionPane**.WARNING_MESSAGE *// this is a warning message*

you can also use JOptionPane for dialog boxes for input, and assign them to variables:

name= **JOptionPane**.showInputDialog ( "put your message here" );

**Example:**

```
import javax.swing.JOptionPane;
public class JOptionPane Example{
public static void main(String[] args) {
	String breadth = JOptionPane.showInputDialog("Rectangle Breadth");
	String height = JOptionPane.showInputDialog("Rectangle Height");
	int area = Integer.parseInt(breadth) * Integer.parseInt(height);
	JOptionPane.showMessageDialog(null, "Area=" + area, "",
JOptionPane.WARNING_MESSAGE);
	System.exit(0);

	}
}
```

## Add action listener that listens to multiple buttons

```
import javax.swing.JFrame;
import javax.swing.JButton;
import java.awt.Color;
import java.awt.event.ActionListener;
import java.awt.event.ActionEvent;
import java.awt.FlowLayout;

public class MultiButtons implements ActionListener {
	JButton btn[] = new JButton[3];
	String colorName[]={"RED","BLUE","GREEN"};

	Color color[] = {Color.red, Color.blue, Color.green};
	JFrame f=new JFrame();

	// Constructor
	MultiButtons (){
		f.setLayout(new FlowLayout());
		// create button and add ActionListener for each button.
		for (int i=0; i<btn.length;i++){
			btn[i]= new JButton(colorName[i]);
			btn[i].addActionListener(this);
			f.add(btn[i]);
		}

		f.setVisible(true);
		f.setSize(300, 200);
	}
	public void actionPerformed(ActionEvent e){
		// the ActionListener monitoring 3 buttons.
		for (int i=0; i<btn.length;i++)
			// the button[i] is being pressed
			if (btn[i]== e.getSource()){
				f.getContentPane().setBackground(color[i]);
				break;
			}
	}
	public static void main(String[] args){

		MultiButtons mb=new MultiButtons ();


	}

}


import javax.swing.JFrame;
import javax.swing.JButton;
import java.awt.Color;
import java.awt.event.ActionListener;
import java.awt.event.ActionEvent;
import java.awt.FlowLayout;

public class MultButton implements ActionListener{
	JFrame frame=new JFrame();
	JButton btn1 = new JButton("RED");
	JButton btn2 = new JButton("GREEN");
```

```
        JButton btn3 = new JButton("BLUE");

        public MultButton(){ //Constructor
                frame.setSize(400,200);
                frame.setLayout(new FlowLayout());
                frame.add(btn1);
                frame.add(btn2);
                frame.add(btn3);
                frame.setVisible(true);
                btn1.addActionListener(this);
                btn2.addActionListener(this);
                btn3.addActionListener(this);
        }

        public void actionPerformed(ActionEvent e){
                // the ActionListener monitoring 3 buttons.
                if (btn1== e.getSource()){
                        frame.getContentPane().setBackground(Color.RED);
                if (btn2== e.getSource())
                        frame.getContentPane().setBackground(Color.green);
                if (btn3== e.getSource())
                        frame.getContentPane().setBackground(Color.red);
                }
        }
        public static void main(String args[]){
                MultButton mb= new MultButton();

        }
}
```

**JTextArea**

The JTextArea class provides a component that displays multiple lines of text and optionally allows the user to edit the text.

**Constructor**: Creates a text area. When present, the String argument contains the initial text. The int arguments specify the desired width in columns and height in rows, respectively.

JTextArea()

JTextArea(String)

JTextArea(String, int, int)

JTextArea(int, int)

**Sets or obtains the text displayed by the text area.**

void setText(String)

String getText()

**Sets or indicates whether the user can edit the text in the text area.**

void setEditable(boolean)

boolean isEditable()

**Sets or obtains the number of columns displayed by the text area.**

void setColumns(int);

int getColumns()

**Sets or obtains the number of rows displayed by the text area**

void setRows(int);

int getRows()

**Sets whether lines are wrapped if they are too long to fit within the allocated width. By default this property is false and lines are not wrapped.**

int setLineWrap(boolean)

**Sets whether lines can be wrapped at white space (word boundaries) or at any character. By default this property is false, and lines can be wrapped (if line wrapping is turned on) at any character.**

int setWrapStyleWord(boolean)

**Adds the specified text to the end of the text area.**

void append(String)

**Inserts the specified text at the specified position.**

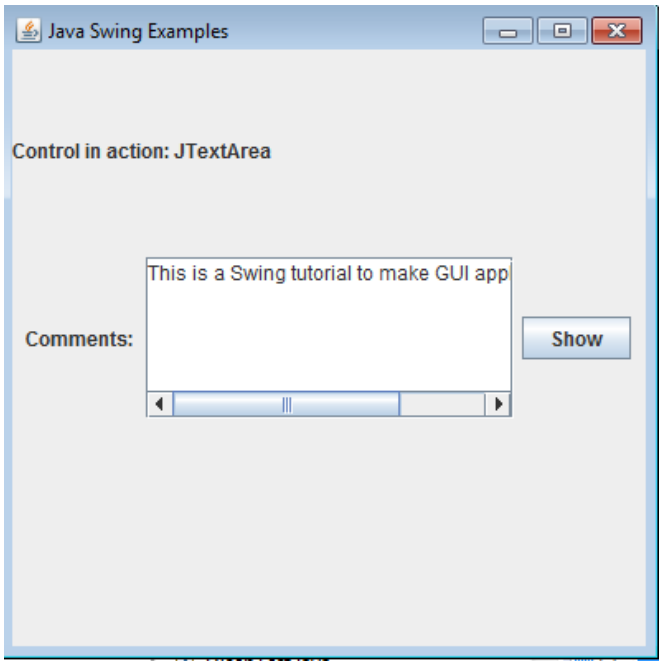
void insert(String, int)

**Replaces the text between the indicated positions with the specified string.**

void replaceRange(String, int, int)

```
import javax.swing.*;
import java.awt.event.*;
import java.awt.*;
public class JTextAreaTest {
  public static void main(String[] args) {
      JFrame mainFrame = new JFrame("Java Swing Examples");
      mainFrame.setSize(400,400);
      mainFrame.setLayout(new GridLayout(3, 1));
      JLabel headerLabel = new JLabel("");
      JLabel statusLabel = new JLabel("");
      headerLabel.setText("Control in action: JTextArea");
      JLabel  commentlabel= new JLabel("Comments: ", JLabel.RIGHT);
      JTextArea commentTextArea =  new JTextArea("This is a Swing tutorial " +"to make GUI
application in Java.",5,20);
```

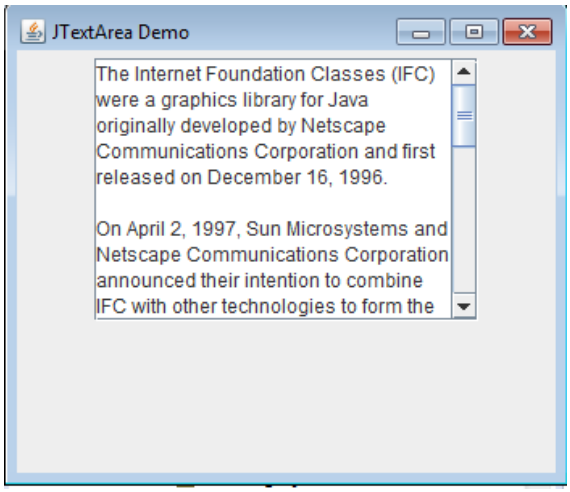
```
        JScrollPane scrollPane = new JScrollPane(commentTextArea);
        JButton showButton = new JButton("Show");
        showButton.addActionListener(new ActionListener() {
           public void actionPerformed(ActionEvent e) {
              statusLabel.setText( commentTextArea.getText());
           }
        });
        JPanel controlPanel = new JPanel();
        controlPanel.setLayout(new FlowLayout());
        controlPanel.add(commentlabel);
        controlPanel.add(scrollPane);
        controlPanel.add(showButton);
        mainFrame.add(headerLabel);
        mainFrame.add(controlPanel);
        mainFrame.add(statusLabel);
        mainFrame.setVisible(true);
         }
   }
```




```
public class TextAreaDemo {
 public static void main(String[] arg){
 JFrame mainFrame = new JFrame("Java Swing Examples");
 mainFrame.setSize(400,400);
 JTextArea sampleTextArea = new JTextArea ("Test");
```

```
 // JScrollPane sampleScrollPane = new JScrollPane
(sampleTextArea,JScrollPane.VERTICAL_SCROLLBAR_AS_NEEDED,JScrollPane.HORIZONTAL_SCROLLBAR_ALWAYS);
   JScrollPane sampleScrollPane = new JScrollPane
   (sampleTextArea,JScrollPane.VERTICAL_SCROLLBAR_ALWAYS,JScrollPane.HORIZONTAL_SCROLLBAR_ALWAYS);
   mainFrame.add(sampleScrollPane);
   mainFrame.setVisible(true);
      }
   }
   public class TextAreaExample {
       public static void main(String[] arg) {
              JPanel totalGUI = new JPanel();
              // This is the story we took from Wikipedia.
                String story = "The Internet Foundation Classes (IFC) were a graphics "
                + "library for Java originally developed by Netscape Communications "
                + "Corporation and first released on December 16, 1996.\n\n"
                + "On April 2, 1997, Sun Microsystems and Netscape Communications"
                + " Corporation announced their intention to combine IFC with other"
                + " technologies to form the Java Foundation Classes. In addition "
                + "to the components originally provided by IFC, Swing introduced "
                + "a mechanism that allowed the look and feel of every component "
                + "in an application to be altered without making substantial "
                + "changes to the application code. The introduction of support "
                + "for a pluggable look and feel allowed Swing components to "
                + "emulate the appearance of native components while still "
                + "retaining the benefits of platform independence. This feature "
                + "also makes it easy to have an individual application's appearance "
                + "look very different from other native programs.\n\n"
                + "Originally distributed as a separately downloadable library, "
                + "Swing has been included as part of the Java Standard Edition "
                + "since release 1.2. The Swing classes are contained in the "
                + "javax.swing package hierarchy.";

              // We create the TextArea and pass the story in as an argument.
              // We also set it to be non-editable, and the line and word wraps set to
              // true.
              JTextArea storyArea = new JTextArea(story, 10, 20);
              storyArea.setEditable(false);
              storyArea.setLineWrap(true);
              storyArea.setWrapStyleWord(true);

       // We create the ScrollPane and instantiate it with the TextArea as an argument
       // along with two constants that define the behavior of the scrollbars.
              JScrollPane area = new
   JScrollPane(storyArea,JScrollPane.VERTICAL_SCROLLBAR_AS_NEEDED,JScrollPane.HORIZONTAL_SCROLLBAR
   _AS_NEEDED);

              totalGUI.add(area);

              JFrame frame = new JFrame("JTextArea Demo");

              frame.setContentPane(totalGUI);

              frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
              frame.setSize(350, 300);
              frame.setVisible(true);
```

```
    }
}
```




**JCheckBox Class**

**JCheckBox** is a Swing component that represents an item, which shows a state of selected or unselected. User can change this state by clicking on the check box of the component.

**JCheckBox()**:Creates an initially unselected check box button with no text, no icon.

**JCheckBox(Icon icon)**:Creates an initially unselected check box with an icon.

**JCheckBox(Icon icon, boolean selected)**:Creates a check box with an icon and specifies whether or not it is initially selected.

**JCheckBox(String text)**: Creates an initially unselected check box with text.

**JCheckBox(String text, boolean selected)**:Creates a check box with text and specifies whether or not it is initially selected.

**Creating a JCheckBox with only text:**

```
JCheckBox checkbox = new JCheckBox("Enable logging");
```

**Creating a JCheckBox with text and selected state:**

```
JCheckBox checkbox = new JCheckBox("Enable logging", true);
```

**The text, icon and selected state can be set later, for example:**

```
JCheckBox checkbox = new JCheckBox();
 checkbox.setText("Enable logging");
checkbox.setSelected(true);
checkbox.setIcon(icon);
```

**Adding the check box to a container**

A JCheckBox component can be added to a container like JFrame or JPanel:

```
frame.add(checkbox);
panel.add(checkbox);
```

**Setting state for the check box:**

```
checkbox.setSelected(true);
checkbox.setSelected(false);
```

**Getting state of the check box:**

```
if (checkbox.isSelected()) {
   // selected, do something...
} else {
   // un-selected, do something else...
 }
```

**Adding event listeners**

```
checkbox.addActionListener(new ActionListener() {
public void actionPerformed(ActionEvent event) {
JCheckBox cb = (JCheckBox) event.getSource();
 if (cb.isSelected()) {
          // do something if check box is selected
} else {
          // check box is unselected, do something else
       }
   }
});
```

**Customizing appearance**

**Setting tooltip text:**

```
checkbox.setToolTipText("If enabled, write debug in information to log files.");
```

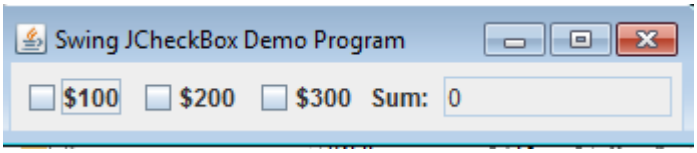
**Setting font style, background color and foreground color:**

checkbox.setFont(new java.awt.Font("Arial", Font.BOLD, 14));

checkbox.setBackground(Color.BLUE);

checkbox.setForeground(Color.YELLOW);

**Example:**




```
import java.awt.FlowLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.JCheckBox;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JTextField;
public class TestCheckBox {
    JFrame frame = new JFrame("Swing JCheckBox Demo Program");
    JCheckBox checkboxOne = new JCheckBox("$100");
    JCheckBox checkboxTwo = new JCheckBox("$200");
    JCheckBox checkboxThree = new JCheckBox("$300");
    // a int sum = 0; label and a text field to display sum
    JLabel labelSum = new JLabel("Sum: ");
    JTextField textFieldSum = new JTextField(10);
    int sum = 0; // sum of 3 numbers
    TestCheckBox() {
          frame.setLayout(new FlowLayout());
          // add the check boxes to this frame
          frame.add(checkboxOne);
          frame.add(checkboxTwo);
          frame.add(checkboxThree);
          frame.add(labelSum);
          textFieldSum.setEditable(false);
          frame.add(textFieldSum);
          // add action listener for the check boxes
          ActionListener actionListener = new ActionListener() {
                public void actionPerformed(ActionEvent event) {
                JCheckBox checkbox = (JCheckBox)event.getSource();
                      if (checkbox.isSelected()) {
                            if (checkbox == checkboxOne) {
                                  sum += 100;
                            } else if (checkbox == checkboxTwo) {
                                  sum += 200;
                            } else if (checkbox == checkboxThree) {
                                  sum += 300;
                            }
                      } else {
                            if (checkbox == checkboxOne) {
                                  sum -= 100;
                            } else if (checkbox == checkboxTwo) {
```

```
                                sum -= 200;
                            } else if (checkbox == checkboxThree) {
                                sum -= 300;
                            }
                        }
                        textFieldSum.setText(String.valueOf(sum));
                    }
                };
                checkboxOne.addActionListener(actionListener);
                checkboxTwo.addActionListener(actionListener);
                checkboxThree.addActionListener(actionListener);
                frame.pack();
                frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
                frame.setVisible(true);
        }
        public static void main(String[] args) {
                new TestCheckBox();

        }
}
```

### JRadioButton Class

The **JRadioButton** class is used to create radio buttons. A set of radio buttons can be associated as a group in which only one button at a time can be selected.

### Common JRadioButton constructors and methods

rb =**new JRadioButton**("label") //Creates unselected radio button.

b = rb.**isSelected**(); //Returns true if that button is selected.

rb.**setSelected**(b); //Sets selected status to b (true/false).

rb.**addActionListener**(actionListener);//Action listener called when button is selected.

### Common ButtonGroup constructors and methods

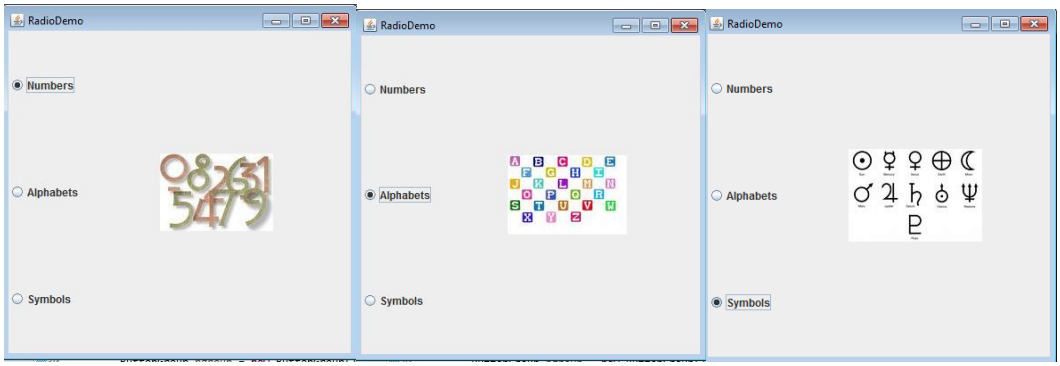
The most common method used for a button group is add(), but you might sometimes want to deselect all of them. Assume:

bg =new ButtonGroup();// Creates a radio button group.

bgroup.add(rb); //Adds a button to a radio button group.

bgroup.clearSelection();//Sets all radio buttons to the unselected state.

**Example**




```
import java.awt.BorderLayout;
import java.awt.GridLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.*;

public class RadioButtonDemo {
    public static void main(String[] args) {
            // ... Create the buttons.
            JFrame frame = new JFrame("RadioDemo");
            JRadioButton numbers = new JRadioButton("Numbers", true);
            JRadioButton alphabets = new JRadioButton("Alphabets");
            JRadioButton symbols = new JRadioButton("Symbols");
            JLabel jlbPicture;
            // Set up the picture label
            jlbPicture = new JLabel(new ImageIcon("" + "numbers" + ".jpg"));

            ActionListener myListener = new ActionListener() {
                  public void actionPerformed(ActionEvent event) {
                        jlbPicture.setIcon(new ImageIcon("" + event.getActionCommand() + ".jpg")
                  }
            };
            numbers.addActionListener(myListener);
            alphabets.addActionListener(myListener);
            symbols.addActionListener(myListener);

            // ... Create a button group and add the buttons.
            ButtonGroup bgroup = new ButtonGroup();
            bgroup.add(numbers);
            bgroup.add(alphabets);
            bgroup.add(symbols);


            JPanel radioPanel = new JPanel();
            radioPanel.setLayout(new GridLayout(0, 1));
            radioPanel.add(numbers);
            radioPanel.add(alphabets);
```

```
        radioPanel.add(symbols);
        frame.add(radioPanel, BorderLayout.WEST);
        frame.add(jlbPicture, BorderLayout.CENTER);
        frame.setSize(400, 400);
        frame.setVisible(true);

    }
}
```

## Java I/O

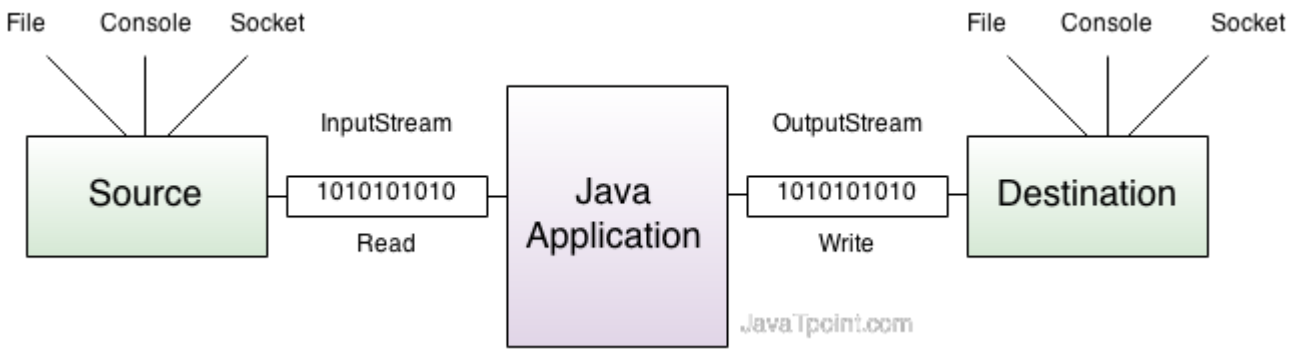
**Java I/O** (Input and Output) is used to process the input and produce the output based on the input.

Java uses the concept of stream to make I/O operation fast. The java.io package contains all the classes required for input and output operations.

### Stream

A stream can be defined as a sequence of data. There are two kinds of streams:

- **InputStream:** the InputStream is used to read data from a source, it may be a file, an array, peripheral device or socket.
- **OutPutStream:** the OutputStream is used for writing data to a destination, it may be a file, an array, peripheral device or socket.




## OutputStream class

OutputStream class is an abstract class. It is the superclass of all classes representing an output stream of bytes. An output stream accepts output bytes and sends them to some sink.

## Commonly used methods of OutputStream class

| Method | Description |
|---|---|
| public void write(int)throws IOException | is used to write a byte to the current output stream. |
| public void write(byte[])throws IOException | is used to write an array of byte to the current output stream. |
| **public void flush()throws IOException** | flushes the current output stream. |
| public void close()throws IOException | is used to close the current output stream. |

### InputStream class

InputStream class is an abstract class.It is the superclass of all classes representing an input stream of bytes.

### Commonly used methods of InputStream class

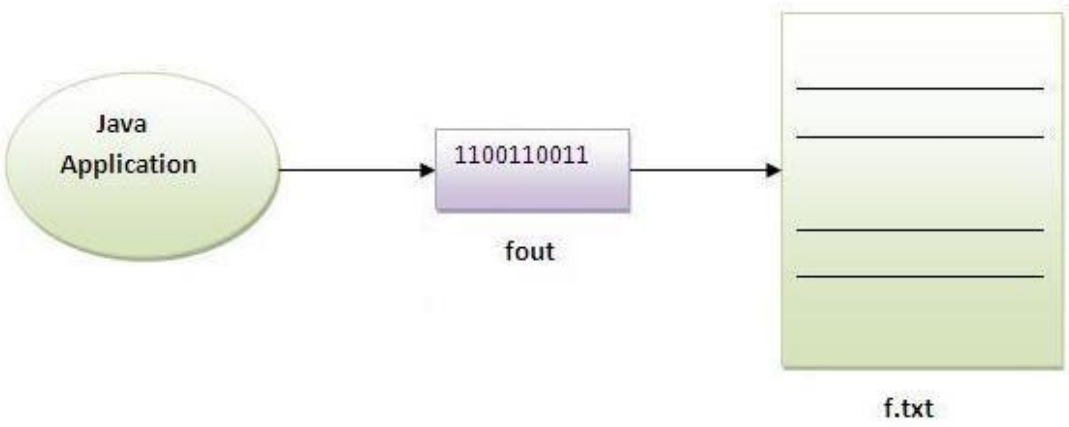
| Method | Description |
|---|---|
| public abstract int read()throws IOException: | reads the next byte of data from the input stream. It returns -1 at the end of file. |
| public int available()throws IOException: | returns an estimate of the number of bytes that can be read from the current input stream. |
| public void close()throws IOException: | is used to close the current input stream. |

Java byte streams are used to perform input and output of 8-bit bytes. However, there are many classes related to byte streams but the most frequently used classes are **FileInputStream** and **FileOutputStream**.

Java **FileOutputStream** class: is an output stream for writing data to a file.

**Example**

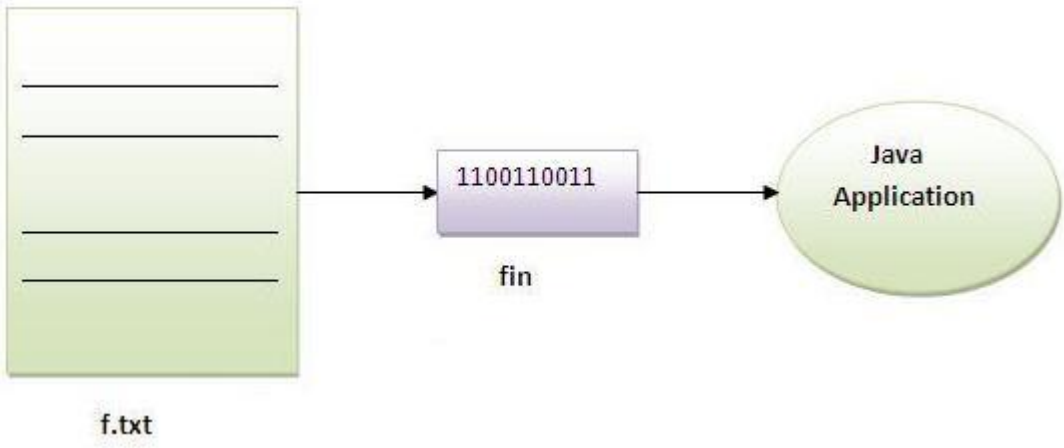
```
import java.io. FileOutputStream;
public class FileTest {
	public static void main(String args[]) {
		try {
			FileOutputStream fout = new FileOutputStream("f.txt");
			String s = "Sachin Tendulkar is my favourite player";
			byte b[] = s.getBytes();// converting string into byte array
			fout.write(b);
			fout.close();
			System.out.println("success...");
		} catch (Exception e) {
			System.out.println(e);
		}
	}
}
```




Java **FileInputStream** class: obtains input bytes from a file. It is used for reading streams of raw bytes such as image data.

**Example**

```
import java.io.FileInputStream;
public class SimpleRead {
        public static void main(String args[]) {
                try {
                        FileInputStream fin = new FileInputStream("f.txt");
                        int i = fin.read();
                        while (i!= -1) {
                                System.out.print((char) i);
                                i = fin.read();
                        }
                        fin.close();
                } catch (Exception e) {
                        System.out.println(e);
                }
        }
}
```




```
import java.io.FileInputStream;
import java.io. FileOutputStream;
import java.io.IOException;
public class FileStreamTest{
        public static void main(String args[]) {
                try {
                        byte bWrite[] = { 2, 21, 3, 40, 5 };
                        FileOutputStream os = new FileOutputStream("test.txt");
                        for (int x = 0; x < bWrite.length; x++) {
                                os.write(bWrite[x]); // writes the bytes

                        }
                        os.close();
                        FileInputStream is = new FileInputStream("test.txt");
                        int size = is.available();

                        for (int i = 0; i < size; i++) {
                                System.out.print(is.read() + "  ");
                        }
                        is.close();
                } catch (IOException e) {
                        System.out.print("Exception");
                }
        }
}
```

## Java BufferedOutputStream and BufferedInputStream

Java BufferedOutputStream class uses an internal buffer to store data. It adds more efficiency than to write data directly into a stream. So, it makes the performance fast.

**Example** of BufferedOutputStream class:

In this example, we are writing the textual information in the BufferedOutputStream object which is connected to the FileOutputStream object. The flush() flushes the data of one stream and send it into another. It is required if you have connected the one stream with another.

```
import java.io.BufferedOutputStream;
import java.io.FileOutputStream;
class FileTest {
        public static void main(String args[]) throws Exception {
                FileOutputStream fout = new FileOutputStream("f1.txt");
                BufferedOutputStream bout = new BufferedOutputStream(fout);
                String s = "Sachin is my favourite player";
                byte b[] = s.getBytes();
                bout.write(b);
                bout.flush();
                bout.close();
                fout.close();
                System.out.println("success");
        }
}
```

**Java BufferedInputStream class** is used to read information from stream. It internally uses buffer mechanism to make the performance fast.

**Example of Java BufferedInputStream**

```
import java.io.*;
class SimpleRead {
        public static void main(String args[]) {
                try {
                        FileInputStream fin = new FileInputStream("f1.txt");
                        BufferedInputStream bin = new BufferedInputStream(fin);
                        int i;
                        while ((i = bin.read()) != -1) {
                                System.out.print((char) i);
                        }
                        bin.close();
                        fin.close();
                } catch (Exception e) {
                        System.out.println(e);
                }
        }}
```

**Character Streams**

Java **Byte** streams are used to perform input and output of 8-bit bytes, where as Java **Character** streams are used to perform input and output for 16-bit unicode. Though there are many classes related to character streams but the most frequently used classes are , **FileReader** and **FileWriter.**

Java **FileWriter** class is used to write character-oriented data to the file.

Constructors of FileWriter class

| Constructor | Description |
|---|---|
| FileWriter(String file) | creates a new file. It gets file name in string. |
| FileWriter(File file) | creates a new file. It gets file name in File object. |

Methods of FileWriter class

| Method | Description |
|---|---|
| public void write(String text) | writes the string into FileWriter. |
| public void write(char c) | writes the char into FileWriter. |
| public void write(char[] c) | writes char array into FileWriter. |
| public void flush() | flushes the data of FileWriter. |
| public void close() | closes FileWriter. |

**Example**

```
import java.io. FileWriter;
public class SimpleWrite {
        public static void main(String args[]) {
                try {
                        FileWriter fw = new FileWriter("abc.txt");
                        fw.write("my name is sachin");
                        fw.close();
                } catch (Exception e) {
                        System.out.println(e);
                }
                System.out.println("success");
        }
}
```

Java **FileReader** class is used to read data from the file. It returns data in byte format like FileInputStream class.

Constructors of FileWriter class

| Constructor | Description |
|---|---|
| FileReader(String file) | It gets filename in string. It opens the given file in read mode. If file doesn't exist, it throws FileNotFoundException. |
| FileReader(File file) | It gets filename in file instance. It opens the given file in read mode. If file doesn't exist, it throws FileNotFoundException. |

Methods of FileReader class

| Method | Description |
|---|---|
| public int read() | returns a character in ASCII form. It returns -1 at the end of file. |
| public void close() | closes FileReader. |

**Example**

```
import java.io. FileReader;
public class SimpleRead{
        public static void main(String args[]) throws Exception{
         FileReader fr=new FileReader("abc.txt");
         int i =fr.read();
         while((i!=-1)
         System.out.print((char)i);
         i =fr.read();
         fr.close();
        }

}
```

**Example** : BufferedReader

```
public class BufferedReaderExample {
   public static void main(String args[])throws Exception{
      FileReader fr=new FileReader("testout.txt");
      BufferedReader br=new BufferedReader(fr);
      int i;
      while((i=br.read())!=-1){
      System.out.print((char)i);
      }
      br.close();
```

```
        fr.close();
    }
}
```

**Example** : BufferedWriter

```
public class WriteToFileExample2 {

        public static void main(String[] args) {

                try{
                        FileWriter fw=new FileWriter("f.txt");
                        BufferedWriter bw = new BufferedWriter(fw);
                        String content = "This is the content to write into file\n";
                        bw.write(content);
                        bw.close();
                        fw.close();
                        System.out.println("Done");

                } catch (IOException e) {

                        e.printStackTrace();
                }
        }

}
```

**Java program to find number of words and lines in a file:**

```
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;

public class WordLineCount {
public static void main(String[] args)
  {
        BufferedReader reader = null;
        //Initializing wordCount and lineCount to 0
        int wordCount = 0;
        int lineCount = 0;
        try
        {
                //Creating BufferedReader object
                reader = new BufferedReader(new FileReader("file1.txt"));

                //Reading the first line into currentLine
                String currentLine = reader.readLine();
                while (currentLine != null)
                {
                lineCount++;//Updating the lineCount
                //Getting number of words in currentLine
                String[] words = currentLine.split(" ");
                wordCount = wordCount + words.length; //Updating the wordCount
                //Reading next line into currentLine
```

```
                currentLine = reader.readLine();
                }
                //Printing wordCount and lineCount
                System.out.println("Number Of Words In A File : "+wordCount);
                System.out.println("Number Of Lines In A File : "+lineCount);
        }
        catch (IOException e)
        {
                e.printStackTrace();
        }
  }
}
```

**Java program to find number of words and lines in a file using read method and testing ASCII code of newline and space:**

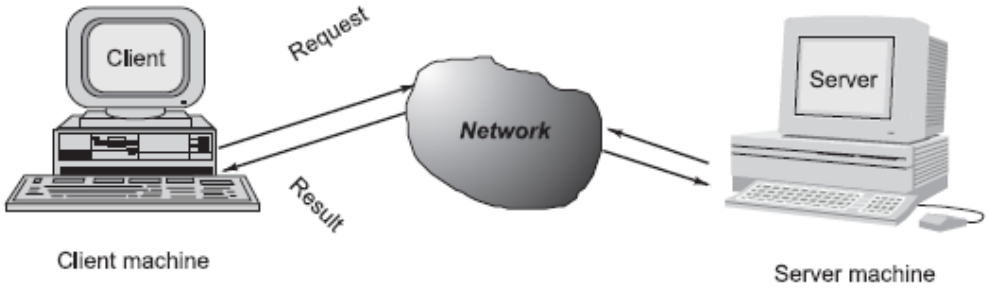
```
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;

public class WordLineCount {
public static void main(String[] args)
   {
       BufferedReader reader = null;
       //Initializing wordCount and lineCount to 0
       int wordCount = 0;
       int lineCount = 0;
       try
       {
               //Creating BufferedReader object
       reader = new BufferedReader(new FileReader("file1.txt"));

               int i=reader.read();

               while ( i!=-1)
               {
                       if (i==10) //ASCII code of newline is 10
                          lineCount++;

                       if (i==32)
                           wordCount++;

                       i=reader.read();
               }

               wordCount=wordCount+lineCount-1;
               System.out.println("Number Of Words In A File : "+wordCount);
               System.out.println("Number Of Lines In A File : "+lineCount);
       }
       catch (IOException e)
       {
               e.printStackTrace();
       }


   }
}
```

## Text File Input with the Scanner Class

```
import java.util.Scanner;
import java.io.File;
import java.io.IOException;

public class trytest {
	public static void main(String args[]) throws IOException{
		Scanner reader = new Scanner(new File("ints.txt"));
		while (reader.hasNext()){
			int i = reader.nextInt();
			System.out.println(i);
		}
		reader.close();
		reader = new Scanner(new File("lines.txt"));
		while (reader.hasNext()){
			String str = reader.nextLine();
			System.out.println(str);
		}

		reader.close();
	}
}
```

**Client/Server Communication**

At a basic level, network-based systems consist of a server, client, and a media for communication as shown in the following figure. A computer running a program that makes a request for services is called client machine. A computer running a program that offers requested services from one or more clients is called server machine. The media for communication can be wired or wireless network.




**Socket Programming and Java.net Class**

**Sockets** provide an interface for programming networks at the transport layer. Network communication using Sockets is very much similar to performing file I/O. In fact, socket handle is treated like file handle. The streams used in file I/O operation are also applicable to socket-based I/O.

A server (program) runs on a specific computer and has a socket that is bound to a specific port. The server listens to the socket for a client to make a connection request. If everything goes well, the server accepts the connection


[a]: a client making a connection request to the server


A **socket** is an endpoint of a two-way communication link between two programs running on the network.

Socket is bound to a port number so that the TCP layer can identify the application that data is destined to be sent. Java provides a set of classes, defined in a package called **java.net**, to enable the rapid development of network applications.

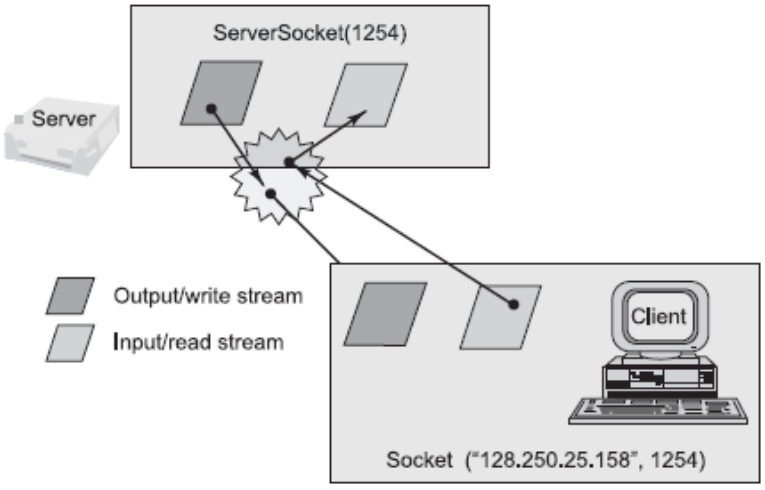
**TCP/IP Socket Programming**

The two key classes from the java.net package used in creation of server and client programs are:

**ServerSocket**

**Socket**

A server program creates a specific type of socket that is used to listen for client requests (server socket). In the case of a connection request, the program creates a new socket through which it will exchange data with the client using input and output streams. The socket abstraction is very similar to the file concept: developers have to open a socket, perform I/O, and close it.




***A simple Server Program in Java***

The steps for creating a simple server program are:

1. Open the Server Socket:

```
ServerSocket server = new ServerSocket( PORT );
```

2. Wait for the Client Request:

```
Socket client = server.accept();
```

3. Create I/O streams for communicating to the client

```
DataInputStream is = new DataInputStream(client.getInputStream());
DataOutputStream os = new DataOutputStream(client.getOutputStream());
```

4. Perform communication with client

**Receive from client**: String line = is.readUTF()**;**

**Send to client**: os. writeUTF("Hello");

5. Close socket:

client.close();

***A simple Client Program in Java***

The steps for creating a simple client program are:

1. Create a Socket Object:

Socket client = new Socket(server, port_id);

2. Create I/O streams for communicating with the server.

is = new DataInputStream(client.getInputStream());

os = new DataOutputStream(client.getOutputStream());

3. Perform I/O or communication with the server:

**Receive data from the server**: String line = is. readUTF();

**Send data to the server**: os.writeUTF (“Hello\n”);

4. Close the socket when done:

client.close();

An example program illustrating establishment of connection to a server and then reading a message sent by the server and displaying it on the console is given below:

```
// SimpleServer.java: A simple server program.
import java.net.*;
import java.io.*;

public class SimpleServer {
        public static void main(String args[]) throws IOException {
                // Register service on port 1254
                ServerSocket s = new ServerSocket(1254);
                System.out.println("Server started: " + s);
                System.out.println("Waiting for a client ...");
                Socket s1 = s.accept(); // Wait and accept a connection
                System.out.println("Client accepted: " + s1);
                // Get a communication stream associated with the socket
                OutputStream s1out = s1.getOutputStream();
                DataOutputStream dos = new DataOutputStream(s1out);
                // Send a string!
                dos.writeUTF("Hi there");
                // Close the connection, but not the server socket

                dos.close();
                s1out.close();
                s1.close();

        }
}
```

**// SimpleClient.java: A simple client program.**

```
import java.net.*;
import java.io.*;
import javax.swing.JOptionPane;

public class SimpleClient {
        public static void main(String args[]) throws IOException {
                // Open your connection to a server, at port 1254
                Socket s1 = new Socket("localhost", 1254);
                System.out.println("Connected: " + s1);
                // Get an input file handle from the socket and read the input
                InputStream s1In = s1.getInputStream();
                DataInputStream dis = new DataInputStream(s1In);
                String answer = new String(dis.readUTF());
                // System.out.println(answer);
                JOptionPane.showMessageDialog(null, answer);
                // When done, just close the connection and exit

                s1.close();
                dis.close();
                s1In.close();

        }
}
```

# Java ArrayList

The ArrayList class is a resizable array, which can be found in the java.util package.

The difference between a built-in array and an ArrayList in Java, is that the size of an array cannot be modified. While elements can be added and removed from an ArrayList whenever you want. The syntax is also slightly different:

**Example**

Create an ArrayList object called **cars** that will store strings:

```
ArrayList<String> cars = new ArrayList<String>(); // Create an ArrayList object
```

**Add Items**

The ArrayList class has many useful methods. For example, to add elements to the ArrayList, use the add() method

Example

```
import java.util.ArrayList;

public class Main {

  public static void main(String[] args) {

    ArrayList<String> cars = new ArrayList<String>();

    cars.add("Volvo");

    cars.add("BMW");

    cars.add("Ford");

    cars.add("Mazda");

    System.out.println(cars);

  }

}
```

**Access an Item**

To access an element in the ArrayList, use the get() method and refer to the index number:

Example

```
import java.util.ArrayList;

public class Main {

  public static void main(String[] args) {

    ArrayList<String> cars = new ArrayList<String>();

    cars.add("Volvo");

    cars.add("BMW");

    cars.add("Ford");
```

```
    cars.add("Mazda");

    System.out.println(cars.get(0));

  }

}
```

**Change an Item**

To modify an element, use the set() method and refer to the index number:

Example

```
import java.util.ArrayList;
public class Main {
  public static void main(String[] args) {
    ArrayList<String> cars = new ArrayList<String>();
    cars.add("Volvo");
    cars.add("BMW");
    cars.add("Ford");
    cars.add("Mazda");
    cars.set(0, "Opel");
    System.out.println(cars);
  }
}
```

**Remove an Item**

To remove an element, use the remove() method and refer to the index number:

Example

```
import java.util.ArrayList;

public class Main {

  public static void main(String[] args) {

    ArrayList<String> cars = new ArrayList<String>();

    cars.add("Volvo");

    cars.add("BMW");

    cars.add("Ford");

    cars.add("Mazda");

    cars.remove(0);

    System.out.println(cars);

  }

}
```

**To remove all the elements in the ArrayList, use the clear() method:**

Example

```
import java.util.ArrayList;
public class Main {
  public static void main(String[] args) {
    ArrayList<String> cars = new ArrayList<String>();
    cars.add("Volvo");
    cars.add("BMW");
    cars.add("Ford");
    cars.add("Mazda");
    cars.clear();
    System.out.println(cars);
  }}
```

**ArrayList Size**

To find out how many elements an ArrayList have, use the size method:

Example

```
cars.size();
```

**Example**

```
import java.util.ArrayList;
public class Main {
  public static void main(String[] args) {
    ArrayList<String> cars = new ArrayList<String>();
    cars.add("Volvo");
    cars.add("BMW");
    cars.add("Ford");
    cars.add("Mazda");
    System.out.println(cars.size());
  } }
```

**Loop Through an ArrayList**

Loop through the elements of an ArrayList with a for loop, and use the size() method to specify how many times the loop should run:

Example

```
import java.util.ArrayList;
public class Main {
  public static void main(String[] args) {
    ArrayList<String> cars = new ArrayList<String>();
    cars.add("Volvo");
    cars.add("BMW");
    cars.add("Ford");
    cars.add("Mazda");
    for (int i = 0; i < cars.size(); i++) {
      System.out.println(cars.get(i));
    }
  }
}
```

You can also loop through an ArrayList with the **for-each** loop:

Example

```
import java.util.ArrayList;
public class Main {

  public static void main(String[] args) {

    ArrayList<String> cars = new ArrayList<String>();

    cars.add("Volvo");

    cars.add("BMW");

    cars.add("Ford");

    cars.add("Mazda");

    for (String i : cars) {

      System.out.println(i);

    }

  }

}
```

**Other Types**

Elements in an ArrayList are actually objects. In the examples above, we created elements (objects) of type "String". Remember that a String in Java is an object (not a primitive type). To use other types, such as int, you must specify an equivalent wrapper class: Integer. For other primitive types, use: Boolean for boolean, Character for char, Double for double, etc.:

Example

Create an ArrayList to store numbers (add elements of type Integer):

```
import java.util.ArrayList;

public class Main {

  public static void main(String[] args) {

    ArrayList<Integer> myNumbers = new ArrayList<Integer>();

    myNumbers.add(10);

    myNumbers.add(15);

    myNumbers.add(20);

    myNumbers.add(25);

    for (int i : myNumbers) {

      System.out.println(i);

    }

  }
```

**Sort an ArrayList**

Another useful class in the java.util package is the Collections class, which include the sort() method for sorting lists alphabetically or numerically:

**Example**

```
import java.util.ArrayList;
import java.util.Collections;  // Import the Collections class
public class Main {
  public static void main(String[] args) {
    ArrayList<String> cars = new ArrayList<String>();
    cars.add("Volvo");
    cars.add("BMW");
    cars.add("Ford");
    cars.add("Mazda");
    Collections.sort(cars);  // Sort cars
    for (String i : cars) {
      System.out.println(i);
    }
  }
}
```

Example

```
import java.util.ArrayList;

import java.util.Collections;  // Import the Collections class

public class Main {

  public static void main(String[] args) {

    ArrayList<Integer> myNumbers = new ArrayList<Integer>();

    myNumbers.add(33);

    myNumbers.add(15);

    myNumbers.add(20);

    myNumbers.add(34);

    myNumbers.add(8);

    myNumbers.add(12);

    Collections.sort(myNumbers);  // Sort myNumbers

    for (int i : myNumbers) {

      System.out.println(i);

    }

  }

}
```