Anywhere(WORA).

## Secured

- No explicit pointer
- Programs run inside virtual machine sandbox.




- **Classloader-** adds security by separating the package for the classes of the local file system from those that are imported from network sources.
- **Bytecode Verifier-** checks the code fragments for illegal code that can violate access right to objects.
- **Security Manager-** determines what resources a class can access such as reading and writing to the local disk.

These security are provided by java language. Some security can also be provided by application developer through SSL,JAAS,cryptography etc.

## Robust

Robust simply means strong. Java uses strong memory management. There are lack of pointers that avoids security problem. There is automatic garbage collection in java. There is exception handling and type checking mechanism in java. All these points makes java robust.

### Architecture-neutral

There is no implementation dependent features e.g. size of primitive types is set.

### Portable

We may carry the java bytecode to any platform.

### High-performance

Java is faster than traditional interpretation since byte code is "close" to native code still somewhat slower than a compiled language (e.g., C++)

### Distributed

We can create distributed applications in java. RMI and EJB are used for creating distributed applications. We may access files by calling the methods from any machine on the internet.



### Multi-threaded

A thread is like a separate program, executing concurrently. We can write Java programs that deal with many tasks at once by defining multiple threads. The main advantage of multi-threading is that it shares the same memory. Threads are important for multi-media, Web applications etc.

### Requirement for Hello Java Example

For executing any java program, you need to

- install the JDK if you don't have installed it, download the JDK and install it.
- set path of the jdk/bin directory. http://www.javatpoint.com/how-to-set-path-in-java
- create the java program
- compile and run the java program

### Creating hello java example

Let's create the hello java program:

```
1. class Simple{
2.     public static void main(String args[]){
3.       System.out.println("Hello Java");
```

| | | |
|---|---|---|
| \|\| | 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 Assignment Operators:

There are following assignment operators supported by Java language:

**Show Examples**

| Operator | Description | Example |
|---|---|---|
| = | Simple assignment operater, 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 | C -= A is equivalent to C = C - A |

```
13. System.out.println(a[i]);
14.
15. }}
```

```
Output: 10
        20
        70
        40
        50
```

---

### Declaration, Instantiation and Initialization of Java Array

We can declare, instantiate and initialize the java array together by:

```
1. int a[]={33,3,4,5};//declaration, instantiation and initialization
```

Let's see the simple example to print this array.

```
1. class Testarray1{
2. public static void main(String args[]){
3.
4. int a[]={33,3,4,5};//declaration, instantiation and initialization
5.
6. //printing array
7. for(int i=0;i<a.length;i++)//length is the property of array
8. System.out.println(a[i]);
9.
10. }}
```

```
Output:33
       3
       4
       5
```



### Multidimensional array in java

In such case, data is stored in row and column based index (also known as matrix form).

### Syntax to Declare Multidimensional Array in java

```
1. dataType[][] arrayRefVar; (or)
2. dataType [][]arrayRefVar; (or)
3. dataType arrayRefVar[][]; (or)
4. dataType []arrayRefVar[];
```

JDK 1.5 introduced a new for loop known as foreach loop or enhanced for loop, which enables you to traverse the complete array sequentially without using an index variable.

# Example:

The following code displays all the elements in the array myList:

```
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 (double element: myList) {
         System.out.println(element);
      }
   }
}
```

This would produce the following result:

```
1.9
2.9
3.4
3.5
```





Java has very flexible three looping mechanisms. You can use one of the following three loops:

- while Loop
- do...while Loop
- for Loop

## Class in Java

A class is a group of objects that has common properties. It is a template or blueprint from which objects are created.

A class in java can contain:

- **data member**
- **method**

### Advantage of Method

- Code Reusability
- Code Optimization

### new keyword

The new keyword is used to allocate memory at runtime.

---

### Example of Object and class that maintains the records of students

In this example, we are creating the two objects of Student class and initializing the value to these objects by invoking the insertRecord method on it. Here, we are displaying the state (data) of the objects by invoking the displayInformation method.

```
1.  class Student2{
2.   int rollno;
3.   String name;
4.
5.   void insertRecord(int r, String n){  //method
6.    rollno=r;
7.    name=n;
8.   }
9.
10.  void displayInformation(){System.out.println(rollno+" "+name);}//method
11.
12.  public static void main(String args[]){
13.   Student2 s1=new Student2();
14.   Student2 s2=new Student2();
15.
16.   s1.insertRecord(111,"Karan");
17.   s2.insertRecord(222,"Aryan");
18.
19.   s1.displayInformation();
20.   s2.displayInformation();
21.
22.  }
23. }
```



**Test it Now**

```
111 Karan
222 Aryan
```

- By clone() method
- By factory method etc.

Java - Methods

A Java method is a collection of statements that are grouped together to perform an operation. When you call the System.out.println method, for example, the system actually executes several statements in order to display a message on the console.

Creating Method:

Considering the following example to explain the syntax of a method:

```
public static int funcName(int a, int b) {
 // body
}
```

Here,

- **public static** : modifier.
- **int**: return type
- **funcName**: function name
- **a, b**: formal parameters
- **int a, int b**: list of parameters

Methods are also known as Procedures or Functions:

- **Procedures:** They don't return any value.
- **Functions:** They return value.

Method definition consists of a method header and a method body. The same is shown below:

```
modifier returnType nameOfMethod (Parameter List) {
 // method body
}
```

```
    s3.display();
    }
}
```

```
Output:111 Karan BBDIT
       222 Aryan BBDIT
       333 Sonoo BBDIT
```

### Another example of static method that performs normal calculation

```
//Program to get cube of a given number by static method

class Calculate{
  static int cube(int x){
  return x*x*x;
  }

  public static void main(String args[]){
  int result=Calculate.cube(5);
  System.out.println(result);
  }
}
```

```
Output:125
```



### Restrictions for static method

There are two main restrictions for the static method. They are:

1. The static method can not use non static data member or call non-static method directly.
2. this and super cannot be used in static context.

```
class A{
 int a=40;//non static

 public static void main(String args[]){
  System.out.println(a);
 }
}
```

```
Output:Compile Time Error
```

### Q) why java main method is static?

Ans) because object is not required to call static method if it were non-static method, jvm create object

## ABSTRACT CLASS

1. A class that is declared with abstract keyword, is known as abstract class in java.
2. It can have abstract and non-abstract methods (CONCRETE METHODS-method with body).
3. An abstract class can not be **instantiated** (you are not allowed to create **object** of Abstract class).
4. It needs to be extended and its method implemented. It cannot be instantiated.
5. Abstract methods must be implemented in the child class (if the class is not abstract) otherwise program will throw compilation error.
6. **Abstraction** is a process of hiding the implementation details and showing only functionality to thE USER.

### Ways to achieve Abstaction

There are two ways to achieve abstraction in java

1. Abstract class (0 to 100%)
2. Interface (100%)

EXAMPLE:



```
abstract public class AbstractDemo{
   public void myMethod(){
      System.out.println("Hello");
   }
   abstract public void anotherMethod();
}
public class ConcreteDemo{

   public void anotherMethod() {
        System.out.print("Abstract method");
   }
   public static void main(String args[])
   {
      //Can't create object of abstract class - error!
      AbstractDemo obj = new AbstractDemo();
      obj.display();
   }
}
```

Output:

```
Unresolved compilation problem: Cannot instantiate the type AbstractEx1
```

Note: The class that extends the abstract class, have to implement all the abstract methods of abstract class, else they can be declared abstract in the class as well.

```
15. obj.print();
16.  }
17. }
```

```
Hello
```

As you can see in the above example, Printable and Showable interface have same methods but its implementation is provided by class A, so there is no ambiguity.

# Interface inheritance

A class implements interface but one interface extends another interface .

```
1.  interface Printable{
2.  void print();
3.  }
4.  interface Showable extends Printable{
5.  void show();
6.  }
7.  class Testinterface2 implements Showable{
8.
9.  public void print(){System.out.println("Hello");}
10. public void show(){System.out.println("Welcome");}
11.
12. public static void main(String args[]){
13. Testinterface2 obj = new Testinterface2();
14. obj.print();
15. obj.show();
16.  }
17. }
```

```
Hello
Welcome
```



**KEY POINTS**

1)We can’t instantiate an interface in java.

2) Interface provides complete abstraction as none of its methods can have body. On the other hand, abstract class provides partial abstraction as it can have abstract and concrete(methods with body) methods both.

3) `implements` keyword is used by classes to implement an interface.

4) While providing implementation in class of any method of an interface, it needs to be mentioned as public.

```
}
```

14) Methods with same signature but different return type can't be implemented at a time for two or more interfaces.

```
interface A
{
   public void aaa();
}
interface B
{
   public int aaa();
}

class Central implements A,B
{

   public void aaa() // error
   {
   }
   public int aaa() // error
   {
   }
   public static void main(String arg[])
   {

   }
}
```



15) Variable names conflicts can be resolved by interface name e.g:

```
interface A
{
    int x=10;
}
interface B
{
    int x=100;
}
class Hello implement A,B
{
    public static void Main(String arg[])
    {

       System.out.println(x); // reference to x is ambiguous both variables
are x
       System.out.println(A.x);
       System.out.println(B.x);
    }
}
```

```
22. class Test5{
23. public static void main(String args[]){
24. A a=new M();
25. a.a();
26. a.b();
27. a.c();
28. a.d();
29. }}
```

Output:

```
I am a
I am b
I am c
I am d
```

## INNER CLASS

**Java inner class** or nested class is a class i.e. declared inside the class or interface.

Additionally, it can access all the members of outer class including private data members and methods.



### *Syntax of Inner class*

```
1. class Java_Outer_class{
2.  //code
3.  class Java_Inner_class{
4.    //code
5.  }
6. }
```

### Advantage of java inner classes

There are basically three advantages of inner classes in java. They are as follows:

1) Nested classes represent a special type of relationship that is **it can access all the members (data members and methods) of outer class** including private.

2) Nested classes are used **to develop more readable and maintainable code** because it logically group classes and interfaces in one place only.

3) **Code Optimization**: It requires less code to write.

### Types of Nested classes

You can use multiple catch block with a single try.

# Problem without exception handling

Let's try to understand the problem if we don't use try-catch block.

```
1.  public class Testtrycatch1{
2.    public static void main(String args[]){
3.       int data=50/0;//may throw exception
4.       System.out.println("rest of the code...");
5.  }
6.  }
```

Output:

```
Exception in thread main java.lang.ArithmeticException:/ by zero
```

As displayed in the above example, rest of the code is not executed (in such case, rest of the code... statement is not printed).

There can be 100 lines of code after exception. So all the code after exception will not be executed.



# Solution by exception handling

Let's see the solution of above problem by java try-catch block.

```
1.  public class Testtrycatch2{
2.    public static void main(String args[]){
3.     try{
4.       int data=50/0;
5.     }catch(ArithmeticException e){System.out.println(e);}
6.     System.out.println("rest of the code...");
7.  }
8.  }
```

Output:

```
Exception in thread main java.lang.ArithmeticException:/ by zero
rest of the code...
```

Now, as displayed in the above example, rest of the code is executed i.e. rest of the code... statement is printed.

# Java throw example

```
1. void m(){
2. throw new ArithmeticException("sorry");
3. }
```

# Java throws example

```
1. void m()throws ArithmeticException{
2. //method code
3. }
```

# Java throw and throws example

```
1. void m()throws ArithmeticException{
2. throw new ArithmeticException("sorry");
3. }
```

**The Java throw keyword** is used to explicitly throw an exception.

We can throw either checked or unchecked exception in java by throw keyword

EXAMPLE :

```
public class TestThrow1{
  static void validate(int age){
    if(age<18)
     throw new ArithmeticException("not valid");
    else
     System.out.println("welcome to vote");
  }
  public static void main(String args[]){
     validate(13);
     System.out.println("rest of the code...");
  }
```

# Java final method

If you make any method as final, you cannot override it.

### Example of final method

```
class Bike{
  final void run(){System.out.println("running");}
}
   class Honda extends Bike{
  void run(){System.out.println("running safely with 100kmph");}
     public static void main(String args[]){
  Honda honda= new Honda();
  honda.run();
  }
}
```

Output:Compile Time Error



# Java final class

If you make any class as final, you cannot extend it.

### Example of final class

```
final class Bike{}
  class Honda1 extends Bike{
  void run(){System.out.println("running safely with 100kmph");}
     public static void main(String args[]){
  Honda1 honda= new Honda();
  honda.run();    } }
```

```
Output:70
```

### static blank final variable

A static final variable that is not initialized at the time of declaration is known as static blank final variable. It can be initialized only in static block.

### Example of static blank final variable

```
1.  class A{
2.    static final int data;//static blank final variable
3.    static{ data=50;}
4.    public static void main(String args[]){
5.      System.out.println(A.data);
6.   }
7.  }
```

### What is final parameter?

If you declare any parameter as final, you cannot change the value of it.

```
1.  class Bike11{
2.    int cube(final int n){
3.     n=n+2;//can't be changed as n is final
4.     n*n*n;
5.    }
6.    public static void main(String args[]){
7.     Bike11 b=new Bike11();
8.      b.cube(5);
9.   }
10. }
```

```
Output:Compile Time Error
```



# Java finalize example

```
1.  class FinalizeExample{
2.  public void finalize(){System.out.println("finalize called");}
3.  public static void main(String[] args){
4.  FinalizeExample f1=new FinalizeExample();
5.  FinalizeExample f2=new FinalizeExample();
6.  f1=null;
7.  f2=null;
8.  System.gc();
9.  }}
```