Data Hiding:- our internal data should not go out directly / outside person can’t access our internal data directly. This OOP feature is known as Data Hiding.

After performing validation we should provide data only to the right person.

Ex:- 1. Gmail login we cant read mails without verifying proper password and username.

2.Even though we are valid customers of the bank we can able to access our account information and we can’t access others account information.

How can we achieve Data hiding – by declaring data member (variable) as PRIVATE.

Recommended modifier for data hiding is PRIVATE.

public class Account{

private double balance;

//How user can access data

public double getBalance(){

//validation

return balance;

}

}

Advantage:- Security

IS-A Relationship:-

Also known as inheritance

-Implemented by using “extends” keyword

-The main advantage is Code Resuability

class P {

public void m1(){

SOPln("Parent");

}

}

class C extends P {

public void m2(){

SOPln("Child");

}

}

-Here Parent class has only 1 method but child class has two methods m1 and m2 bcz it's inheriting from parent class.

class Test{

P S V M(Strings args[]){

1. Parent P = new P();

P.m1();

P.m2();

// Parent can call it's methods but it can't access child method. So P.m2() will give Compile Error-Cannot find symbol method m2()

2.Child C = new C();

C.m1();

C.m2();

//Both will be called successfully

3. Parent P1 = new C();

P1.m1();

P1.m2()

//Conclusion 3 so P1.m2() Gives same CE as above

4. Child C1=new P()

//CE: incompatible types found:P required:C

}

}

Conclusion:-

1.Whatever methods Parent has by default available to the child and Hence on the child reference we can both parent and child class methods.

2.Whatever methods child has by default not available to the Parent and hence on the Parent reference we can't call child specific methods.

3.Parent reference can be used to hold child object but using that reference we can't call child specific methods but we can call the methods preset in parent class.

4.Parent reference can be used to hold child object but child reference can't be used to hold Parent object.

Code Resuability in Inheritance Example:-

Suppose we are writing a Loan module without inheritance and we have 3 classes and 300 methods in each so we have to write 900 methods.

But using inheritance we can write common methods in a class and inherit them to the required classes.

Suppose 250 methods in each above classes are common so write them in a Loan class and extends this class to the other 3.Now we have to write only 50 specific methods which are required by class in each class. So total methods we have to write now are 400 only.

1.Total Java API is implemented based on inheritance. The most common methods which are applicable for any java object are defined in Object class and hence every class in java is a child class of Object either directly or indirectly. So that object class method are default available to java class without rewriting due to this object class as root for all java classes.

Object(11 methods by default available to all child classes) class contains string , String buffer , throwable etc.

2. Throwable defines the most common methods which are required for every exception and error classes hence this class acts as root for java exception hierarchy.

Mulitple Inheritance:-

Java don’t support multiple inheritance in classes because it can’t extends more than one class at a time.

Class A extends B,C {} **X**

Note : If our class doesn’t extend any other class then only our class is direct child class of objet.

class A {}

Object 🡨 A . Class A is child of object.

2. If our class extends any other class then our class is indirect child class of object.

class A extends B {}

Object 🡨 B 🡨 A . A is the child of B. B is the child of object. This is MultiLevel Inheritance.

Note: Either directly or indirectly java won’t provide support for inheritance wrt to classes

Why java don’t Support multiple inheritance ?

-Due to the probability of Ambiguity Problem

P1(m1 method) P2(m1 method) and if we extends both classes on child and call m1 it don’t know which m1 to call.

But interface can extends any number of interfaces simultaneously. Hence java provide support for multiple inheritance with respect to interfaces.

interface A {} interface B {} interface C extends A,B {}

Why ambiguity problem won’t be there in interfaces ?

PI1 {m1();} PI2{m2();} 🡨 CI inherits from both m1(); 🡨 implementation class

Parent interface 1 and 2 both contain m1 method but only declarations are there only implementation class implement and writes the code in m1 so no way of ambiguity.

Even though multiple method declarations are available but implementation is unique and hence there is no chance of ambiguity problem in interfaces.

Note : Strictly speaking through interfaces we won’t get any inheritance.

Class A extends A -- A is the child of same class

Class A extends B and class B extends A --A is child of B and B is child of A

This type of inheritance is called cyclic inheritance and it’s not allowed in java.

**IS-A vs HAS-A**

If we want total functionality of a class automatically then we should go for IS-a relationship.

Ex:- complete functionality of person class is required for student class

If we want part of the functionality then we should go for HAS-A relationship.

Ex:- Test class contains 100 method but within demo class we need 2-3 methods.

**Diff bw Interface and abstraction**:-

Interface is 100% abstraction it has definition

|  |  |
| --- | --- |
| **Abstract class** | **Interface** |
| 1) Abstract class can **have abstract and non-abstract** methods. | Interface can have **only abstract** methods. Since Java 8, it can have **default and static methods** also. |
| 2) Abstract class **doesn't support multiple inheritance**. | Interface **supports multiple inheritance**. |
| 3) Abstract class **can have final, non-final, static and non-static variables**. | Interface has **only static and final variables**. |
| 4) Abstract class **can provide the implementation of interface**. | Interface **can't provide the implementation of abstract class**. |
| 5) The **abstract keyword** is used to declare abstract class. | The **interface keyword** is used to declare interface. |
| 6) An **abstract class** can extend another Java class and implement multiple Java interfaces. | An **interface** can extend another Java interface only. |
| 7) An **abstract class** can be extended using keyword "extends". | An **interface** can be implemented using keyword "implements". |

There are two ways to achieve **abstraction** in java

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

 It can have **abstract** methods(methods without body) as well as concrete methods (regular methods with body). ... An **abstract class** can not be instantiated, which means you are not allowed to create an object of it.

**Friend Function**

In object-oriented programming, a **friend function**, that is a "**friend**" of a given **class**, is a **function** that is given the same access as **methods** to private and protected data. A **friend function** is declared by the **class** that is granting access, so **friend functions** are part of the **class** interface, like **methods**.

**Run Without Main**

Yes, **we can execute** a java **program without** a **main** method by using a static block. Static block in Java is a group of statements that gets **executed** only once when the class is loaded into the memory by Java ClassLoader, It is also known as a static initialization block.

**Types of Inheritance**

On the basis of class, there can be **three types of inheritance** in java: single, multilevel and hierarchical. In java programming, multiple and **hybrid inheritance** is supported through interface only.

**We can**'t **use both** the keywords in the **constructor**. In Java there is a rule that this**() and super()** must be first statement in the **constructor**. So **we can**'t **use both** together in a **constructor**. ... **super()** calls the base class **constructor** & this**()** calls the current class **constructor**.

**Why pointers are not used in Java?**

Java doesn’t use pointers because they are unsafe and increases the complexity of the program. Since, Java is known for its simplicity of code, adding the concept of pointers will be contradicting. Moreover, since JVM is responsible for implicit memory allocation, thus in order to avoid direct access to memory by the user,  pointers are discouraged in Java.

The "this" keyword is used as a reference to an instance. Since the **static methods** doesn't have (belong to) any instance **you** cannot use the "this" reference within a **static method**.

**Can we overload static methods?**  
The answer is ‘Yes’. We can have two ore more static methods with same name, but differences in input parameters. For example, consider the following Java program.

**Can we override static method in Java**

No, you cannot override static method in Java because [method overriding](http://java67.blogspot.sg/2012/08/what-is-method-overriding-in-java-example-tutorial.html) is based upon dynamic binding at runtime and static methods are bonded using [static binding](http://javarevisited.blogspot.sg/2012/03/what-is-static-and-dynamic-binding-in.html) at compile time. Though you can declare a method with same name and method signature in sub class which does look like you can override static method in Java but in reality that is method hiding. Java won't resolve method call at runtime and depending upon type of Object which is used to call [static method](http://javarevisited.blogspot.sg/2011/11/static-keyword-method-variable-java.html), corresponding method will be called. It means if you use Parent class's type to call static method, original static will be called from patent class, on ther other hand if you use Child class's type to call static method, method from child class will be called.

Read more: <https://www.java67.com/2012/08/can-we-override-static-method-in-java.html#ixzz6T1CMzEpM>

**What is instance**

An object and an instance are the **same thing**.

Personally I prefer to use the word "instance" when referring to a specific object of a specific type, for example "an instance of type Foo". But when talking about objects in general I would say "objects" rather than "instances".

A reference either refers to a specific object or else it can be a null reference.

They say that they have to create an instance to their application. What does it mean?

They probably mean you have to write something like this:

Foo foo = new Foo();

### Why use clone() method ?

The **clone() method** saves the extra processing task for creating the exact copy of an object. If we perform it by using the new keyword, it will take a lot of processing time to be performed that is why we use object cloning.

Differences between [**Constructors**](https://www.geeksforgeeks.org/constructors-in-java/) and [**Methods**](https://www.geeksforgeeks.org/methods-in-java/):

| **CONSTRUCTORS** | **METHODS** |
| --- | --- |
| A Constructor is a block of code that initializes a newly created object. | A Method is a collection of statements which returns a value upon its execution. |
| A Constructor can be used to initialize an object. | A Method consists of Java code to be executed. |
| A Constructor is invoked implicitly by the system. | A Method is invoked by the programmer. |
| A Constructor is invoked when a object is created using the keyword **new**. | A Method is invoked through method calls. |
| A Constructor doesn’t have a return type. | A Method must have a return type. |
| A Constructor initializes a object that doesn’t exist. | A Method does operations on an already created object. |
| A Constructor’s name must be same as the name of the class. | A Method’s name can be anything. |
| A class can have many Constructors but must not have the same parameters. | A class can have many methods but must not have the same parameters. |
| A Constructor cannot be inherited by subclasses. | A Method can be inherited by subclasses. |

**Overloaded Main**

There can be more than one main method in a single program. But JVM will always calls String[] argument main() method. Other method's will act as a Overloaded method. These overloaded method's we have to call explicitly.

There can be more than one main method in a single program. Others are Overloaded method. This overloaded method works fine under a single main method

public class MainMultiple{

public static void main(String args[]){

main(122);

main('f');

main("hello java");

}

public static void main(int i){

System.out.println("Overloaded main()"+i);

}

public static void main(char i){

System.out.println("Overloaded main()"+i);

}

public static void main(String str){

System.out.println("Overloaded main()"+str);

}

}

## **Java final variable**

If you make any variable as final, you cannot change the value of final variable(It will be constant).

## **Java final method**

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

## **Java final class**

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

### What is blank or uninitialized final variable?

A final variable that is not initialized at the time of declaration is known as blank final variable.

Can we initialize blank final variable?

Yes, but only in constructor. For example:

1. **class** Bike10{
2. **final** **int** speedlimit;//blank final variable
4. Bike10(){
5. speedlimit=70;
6. System.out.println(speedlimit);
7. }

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. }

### Can we declare a constructor final?

No, because constructor is never inherited.