Day-7 JA-111 Batch

Topics

- **Inheritance**
- **super keyword**
- **Method Overriding**
- **The Object class**

Inheritance

Say I have a developer 'Brendon' who is assigned a task of creating a class to represent a Human. Brendon is writing the code

```
class Human{
  int age;
  String name;
  double weight;
  String bloodGroup;

  //Some methods like walk(), eat(), sleep(), run() etc.
}
```

Now another developer 'Fredrick' is assigned with a task to create a class to represent a Student. It is common sense that a student is s Human i.e. all variables and method of Human must be part of Student class also Student class has its own variables and methods

```
class Student{
  int age;
  String name;
  double weight;
  String bloodGroup;

  //Some methods like walk(), eat(), sleep(), run() etc.

  int rollNo;
  float percentage;
  long registrationNumber;

  //Some methods like admission(), result(), attendance() etc.
}
```

Now 'Fredrick' is assigned with a task to create to create a class to represent school student. Again no need to say that a school student is a Human i.e. all variables and method of Human & Student class must be part of SchoolStudent class also SchoolStudent class has its own variables and methods.

```
class SchoolStudent{
  int age;
  String name;
  double weight;
  String bloodGroup;
  //Some methods like walk(), eat(), sleep(), run() etc.

  int rollNo;
  float percentage;
  long registrationNumber;

  //Some methods like admission(), result(), attendance() etc.

  String boardName;
  String uniformColor;

  //Some methods like annulFunction(), activities() etc.
}
```

In the above implementations you have seen that the code of Human class is copied into two classes that are Student and SchoolStudent. Similarly the code of Student class is copied into the SchoolStudent class. This leads to make the code **WET** which in turn leads to difficulty in maintaining and scaling the code.

e.g. Say we want to add one more attribute to Human class that is DNASequence which is of String type so we have to add the same variable to Student and SchoolStudent class also; In future if we want to remove the method run() from Human then we have remove the same code from Student and SchoolStudent class also.

If you create classes PrimarySchoolStudent, SecondarySchoolStudent and SeniorSecondarySchoolStudent then the code is going to be too lenghty to manage. How to handle such kind of situation?

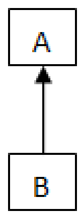
Solution: Inheritance

- Inheritance is a mechanism by which one object acquired properties of another object.
- In enhances code reusability and keep the code DRY that leans to easy maintenance and scalbility of the code.
- It is used to define create specific implementation from a general implementation.
- It is used to define is-a relationship which is Unidirectional.
  E.g. House is a Building. But Building is not a House.

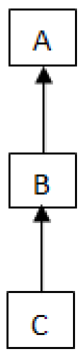
Terminology

Super class/Base class/Parent class: A class from which another class is derived.
Sub class/Derived class/Child class: A class that is derived from a superclass.

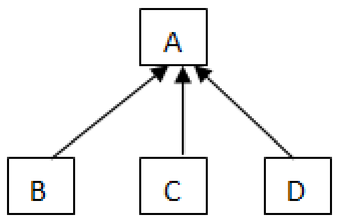
Types of Inheritance



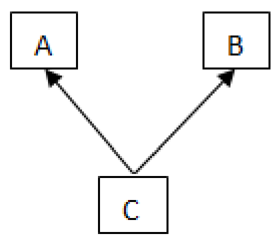
Single level Inheritance



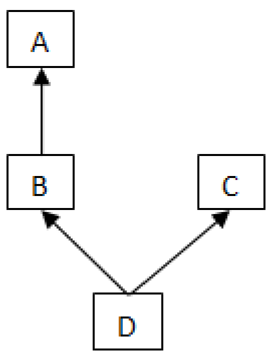
Multi-level Inheritance



Hierarchical Inheritance



Multiple Inheritance



Hybrid Inheritance

General form of class declaration that inherits a superclass is

```
class subclass-name extends superclass-name{
  //body of class
}
```

Here extends is a keyword that denote that we are going to create an specific implementation of super class.

An Example

```
package com.masai.sprint2;
public class A{
  int a;
  int b;

  void setA(int a, int b){
    this.a = a;
    this.b = b;
  }

  void showA(){
    System.out.println("a is " + a);
    System.out.println("b is " + b);
  }
}

package com.masai.sprint2;
public class B extends A{
  int c;

  void setB(int c){
    this.c = c;
  }

  void showB(){
    System.out.println("c is " + c);
  }

  void average(){
    System.out.println("Average is " + (float)(a + b + c) / 3);
  }
}

package com.masai.sprint2;
public class InheritanceDemo {
  public static void main(String[] args) {
    A aOne = new A();
    aOne.setA(100,200);
    aOne.showA();

    System.out.println();

    B bOne = new B();
    bOne.setA(50, 95 );
    bOne.showA();
    bOne.setB(140);
    bOne.showB();

    System.out.println();

    bOne.average();
    //aOne.average();
  }
}
```

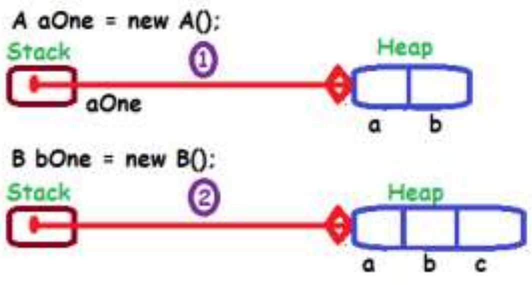
How many variables class B has?
How many methods class B has?

Output

```
a is 100
b is 200

a is 50
b is 95
c is 140

Average is 95.0
```



Tip: To prevent access of instance or class member of a superclass in sub class, simply declare them private. Instance or class members that are declared as private are only accessible in their own classes, not anywhere else including their subclass.

Constructor calling and inheritance

- An object of sub class has all the variables and methods of super class so the object of sub class can be called object of super class.

- All instance members and class members are getting inherited in the sub-class but constructor is not inheritred.
- When an object of subclass is created then constructor of super class as well as of sub class will be called and constructors are called in the order of derivation i.e. they are called from super-class to sub-class.

```
package com.masai.sprint2;
public class A{
  A(){
    System.out.println("Inside constructor of class A");
  }
}

package com.masai.sprint2;
public class B extends A{
  B(){
    System.out.println("Inside constructor of class B");
  }
}

package com.masai.sprint2;
public class C extends B {
  C(){
    System.out.println("Inside constructor of class C");
  }
}

package com.masai.sprint2;
public class Demo{
  public static void main(String[] args) {
    System.out.println("For object of class A");
    A a = new A();
    System.out.println("-=-=-=-=-=-=-=-=-=-=-=-=-=");
    System.out.println("For object of class B");
    B b = new B();
    System.out.println("-=-=-=-=-=-=-=-=-=-=-=-=-=");
    System.out.println("For object of class C");
    C c = new C();
  }
}
```

Output

```
For object of class A
Inside constructor of class A
-=-=-=-=-=-=-=-=-=-=-=-=-=
For object of class B
Inside constructor of class A
Inside constructor of class B
-=-=-=-=-=-=-=-=-=-=-=-=-=
For object of class C
Inside constructor of class A
Inside constructor of class B
Inside constructor of class C
```

- The default constructor (means constructor with no parameter) of super class is called implicitly when object of sub class is created but if super class do not have default constructor (means it has parameterized constructor(s) only) then when object of sub class is created then we have to call super class constructor explicitly using 'super' keyword.
- The super keyword must be the first statement inside the sub class constructor

```
package com.masai.sprint2;
class A{
  A(int i){
    System.out.println("inside constructor of class A");
  }
}

package com.masai.sprint2;
class B extends A{
  B(){
    System.out.println("inside constructor of class B");
  }

  public static void main(String... args){
    B b = new B();
  }
}
```

The above code will produce Compile Time Error because super class (class A) has parameterized constructor only and default constructor of super class is called implicitly but parameterized constructor is not and calling the super class constructor is must. To correct this

```
package com.masai.sprint2;
class A{
  A(int i){
    System.out.println("inside constructor of class A");
  }
}

package com.masai.sprint2;
class B extends A{
  B(){
    super(10);
    System.out.println("inside constructor of class B");
  }

  public static void main(String... args){
    B b = new B();
  }
}
```

Output

```
inside constructor of class A
inside constructor of class B
```

Another use of super keyword is to access hidden instance variable of super class

```
package com.masai.sprint2;
class A{
  int i = 10;
  int j = 20;
}

package com.masai.sprint2;
class B extends A{
  int i = 100;

  void showB() {
    System.out.println("i = " + i);
    System.out.println("j = " + j);
    System.out.println("super.i = " + super.i);
  }
}

package com.masai.sprint2;
class C extends B {
  int i = 1000;
  int j = 2000;

  void showC() {
    System.out.println("i = " + i);
    System.out.println("j = " + j);
    System.out.println("super.i = " + super.i);
    System.out.println("super.j = " + super.j);
    //System.out.println("super.super.i = " + super.super.i);
  }
}

package com.masai.sprint2;
class Demo{
  public static void main(String[] args) {
    B b = new B();
    b.showB();
    System.out.println("-=-=-=-=-=-=-=-=");
    C c = new C();
    c.showC();
  }
}
```

Output

```
i = 100
j = 20
super.i = 10
-=-=-=-=-=-=-=-=
i = 1000
j = 2000
super.i = 100
super.j = 20
```

Method overriding

- If a sub class has a method whose signature (means return-type method-name(para-list)) is same as that of its super class (but not body) then we say that sub class has overridden the method of super class.
- Method overriding is used to redefine (not adding) the behavior of super class means overriding does not add any new method in sub class.
- If overrideen method is called inside the subclass body then version defined in the subclass will be called always and to access the version of super class in the sub class body we have to use super keyword.
- when overriding method in the subclass then use @Override annotation with method signature so that compiler will check if method signature in super class and sub class is same or not. If not same then compiler will report error.

```
package com.masai.sprint2;
class A{
  void show(){
    System.out.println("Inside show of class A");
  }
}

package com.masai.sprint2;
class B extends A{
  void show(){  //class B has overridden the show method of class A
    System.out.println("Inside show of class B");
  }

  void fun(){
    show();
    super.show();
  }

  public static void main(String args){
    A a = new A();
    a.show();
    System.out.println("-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-==");
    B b = new B();
    b.show();
    System.out.println("-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-==");
    b.fun();
  }
}
```

Output:

```
Inside show of class A
-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-==
Inside show of class B
-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-==
Inside show of class B
Inside show of class A
```

A real world example of inheritance

Say we have a class that is used to present price of a product (Say mobile) and the price is having three components that are purchaseCost, cgstPercentage, sgstPercentage and profitPercentage.

```
package com.masai.sprint2;

public class MobilePrice {
  private double purchaseCost;
  private double cgstPercentage;
  private double sgstPercentage;
  private double profitPercentage;

  public MobilePrice(double purchaseCost, double cgstPercentage, double sgstPercentage, double profitPercentage) {
    this.purchaseCost = purchaseCost;
    this.cgstPercentage = cgstPercentage;
    this.sgstPercentage = sgstPercentage;
    this.profitPercentage = profitPercentage;
  }

  public double getPurchaseCost() {
    return purchaseCost;
  }

  public double getSellingPrice() {
    double sellingPrice = purchaseCost + purchaseCost *(cgstPercentage + sgstPercentage + profitPercentage) / 100.0;
    return sellingPrice;
  }
}
```

Say on some selected Luxary Models whose cost is more than 40000/- additional tax that is cess is also imposed and the cess percentage varies state to state. this time instead of making changes in the class MobilePrice it is better to create a subclass of the MobilePrice class.

```
package com.masai.sprint2;

public class LuxaryMobilePrice extends MobilePrice {
  private double cessPercentage;

  public LuxaryMobilePrice(double purchaseCost, double cgstPercentage, double sgstPercentage, double profitPercentage, double cessPercentage) {
    super(purchaseCost, cgstPercentage, sgstPercentage, profitPercentage);
    this.cessPercentage = cessPercentage;
  }

  @Override
  public double getSellingPrice() {
    double sellingPrice = super.getSellingPrice() + (getPurchaseCost() * cessPercentage / 100.0);
    return sellingPrice;
  }
}
```

Now creating a main class

```
package com.masai.sprint2;

public class MobilePriceDemo {
  public static void main(String[] args) {
    MobilePrice mp = new MobilePrice(10000, 7.5, 7.5, 9.5);
    System.out.println("Price of mobile is " + mp.getSellingPrice());

    LuxaryMobilePrice lmp = new LuxaryMobilePrice(45000, 2.5, 3.5, 4.5, 0.5);
    System.out.println("Price of luxary mobile is " + lmp.getSellingPrice());
  }
}
```

Output

```
Price of mobile is 12450.0
Price of luxary mobile is 49950.0
```

The final keyword

It has three uses-

To create named constant

- The variable has to be precede with the final keyword
- If instance variable is declared with final keyword then it can be initialized at the time of declaration or in the constructor but not at both places simulatenously.
- Once the value is provided to final variable (int constructor or declaration) then no change is allowed means you can't write it on LHS of = operator.

Example-1

```
class A{
  final int i;  //this is final, so must be initialized in constructor because
value not provided in declaration
  int j;

  A(){
    i = 10;  //final variable is initialized here
    j = 20;
  }

  void change(){
    //i = 100;  Error because final variable's value cannot be changed.
    j = 200;  //okay
  }
}

Example-2
class A{
  final int i = 10;  //this is final, value is provided in declaration so no
need to assign in constructor now
  int j;

  A(){
    //i = 10;  Error
```

```
      j = 20;
    }

    void change(){
      //i = 100;  Error because final variable's value cannot be changed.
      j = 200;  //okay
    }
  }
```

To prevent method overriding

- If a method is preceded with final keyword then it cannot be overloaded in subclass but it will be inherited.

```
class A{
  final void show(){
    System.out.println("Inside show");
  }
}

class B extends A{
  void show(){   //Error because show is final so can't be overridden in
subclass
    System.out.println("Inside show");
  }

  public static void main(String args[]){
    B b = new B();
    b.show();  //okay
  }
}
```

To prevent inheritance

- If a class is preceded with final keyword then it cannot be inherited.
  e.g. String class is a final class

An Example

```
final class A{}
class B extends A{}  //Error can't subclass a final class
class C extends String{}  //Error can't subclass a final class
```

The Object classes

This is inbuild class that is parent class of all classes in java i.e.

```
class A{}
```

is same as

```
class A extends Object{}
```

All method of Object class are available in all classes of java. The Object class has no variables. Some of the important methods of Object class are-

final Class getClass(): This method returns an object of Class that contains details about the class of the calling object. It has String getName() method that is used to get name of class.

public int hashCode(): The hashcode is an integer value that is calculated on the behalf of the physical address of the object such that the hashcode is stored in the reference variable. From the hashCode, physical address is calculated by JVM to access the object. The mechanism to convert the physical address to the hashcode is confidential.

public boolean equals(Object o): It is used to check equality of the calling object to the parameter object such that by default it uses == operator to check equality i.e. if calling reference variable and parameter reference variable points to same object then it will return true, false otherwise.

public String toString(): This method is used to provide textual representaion of the object so this method is called implicitly when an object is passed to System.out.println() or object is concatenated to any String value. Default implementation of the toString() method returns following textual representaion
class-name@hashcode-in-hex

An Example

```
package com.masai.sprint2;

public class ABC {
  int i;
  int j;

  ABC(){
    i = 10;
    j = 20;
  }

  void show() {
    System.out.println(i + " " + j);
  }
}


package com.masai.sprint2;

public class ObjectClassMethodDemo {
  public static void main(String[] args) {
    ABC a1 = new ABC();
    ABC a2 = new ABC();
    ABC a3 = a2;

    System.out.println(a1.equals(a2));
    System.out.println(a2.equals(a3));
    System.out.println(a1.equals(a3));

    System.out.println();

    System.out.println(a1.hashCode() + " " +
Integer.toHexString(a1.hashCode()));
    System.out.println(a2.hashCode() + " " +
Integer.toHexString(a2.hashCode()));
    System.out.println(a3.hashCode() + " " +
Integer.toHexString(a3.hashCode()));
```

```
    System.out.println();

    System.out.println(a1);
    System.out.println(a2);
    System.out.println(a3.toString());
  }
}
```

Output

```
false
true
false

1579572132 5e265ba4
359023572 156643d4
359023572 156643d4

com.masai.sprint2.ABC@5e265ba4
com.masai.sprint2.ABC@156643d4
com.masai.sprint2.ABC@156643d4
```

Apart from these methods it has following methods also

- **protected Object clone()**: Creates and returns a copy of this object.
- **protected void finalize()**: Called by the garbage collector on an object when garbage collection determines that there are no more references to the object.
- **public void notify()**: Wakes up a single thread that is waiting on this object's monitor.
- **public void notifyAll()**: Wakes up all threads that are waiting on this object's monitor.
- **public void wait()**: Causes the current thread to wait until another thread invokes the notify() method or the notifyAll() method for this object.
- **public void wait(long timeout)**: Causes the current thread to wait until either another thread invokes the notify() method or the notifyAll() method for this object, or a specified amount of time has elapsed.
- **public void wait(long timeout, int nanos)**: Causes the current thread to wait until another thread invokes the notify() method or the notifyAll() method for this object, or some other thread interrupts the current thread, or a certain amount of real time has elapsed.