MODULE - 2

CHAPTER – 3 INHERITANCE

INHERITANCE IN JAVA

- Inheritance in Java is a mechanism in which one object acquires all the properties and behaviors of a parent object.
- The idea behind inheritance in Java is that you can create new classes that are built upon existing classes.
- When you inherit from an existing class, you can reuse methods and attributes of the parent class. Moreover, you can add new methods and attributes in your current class also
- Inheritance represents the **IS-A relationship** which is also known as a *parent-child* relationship.

❖Terms used in Inheritance

Class: A class is a template or blueprint from which objects are created.

Sub Class/Child Class: Subclass is a class which inherits the other class. It is also called a derived class, extended class, or child class.

Super Class/Parent Class: Superclass is the class from where a subclass inherits the features. It is also called a base class or a parent class.

Reusability: As the name specifies, reusability is a mechanism which facilitates you to reuse the attributes and methods of the existing class when you create a new class. We can use the same attributes and methods already defined in the previous class.

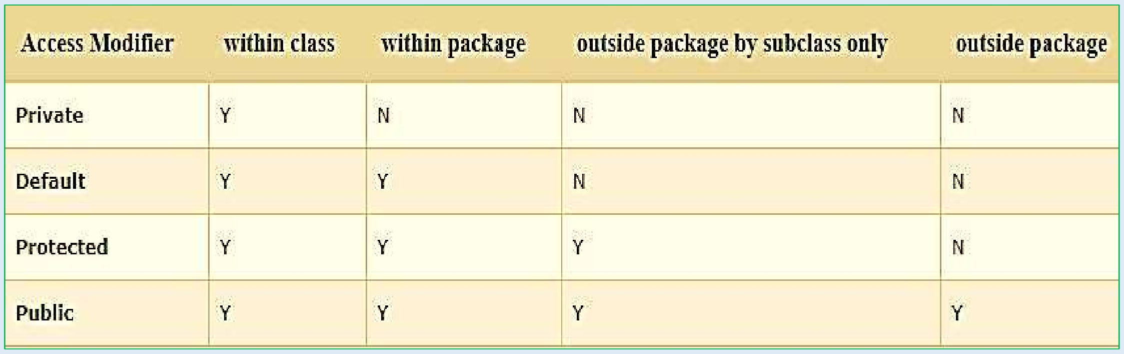
❖Access Modifiers - There are four types of Java access modifiers:

Private: The access level of a private modifier is only within the class. It cannot be accessed from outside the class.

Default: The access level of a default modifier is only within the package. It cannot be accessed from outside the package. If you do not specify any access level, it will be the default.

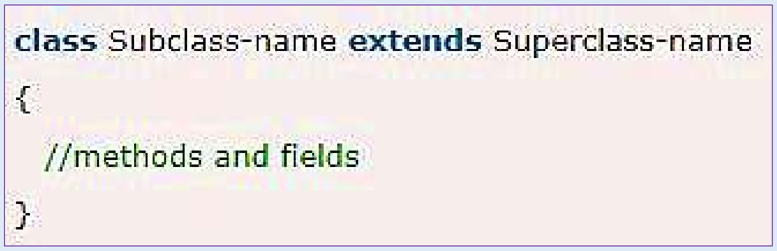
Protected: The access level of a protected modifier is within the package and outside the package through child class. If you do not make the child class, it cannot be accessed from outside the package.

Public: The access level of a public modifier is everywhere. It can be accessed from within the class, outside the class, within the package and outside the package.



Access Modifier	within class	within package	outside package by subclass only	outside package
Private	Y	N	N	N
Default	Y	Y	N	N
Protected	Y	Y	Y	N
Public	Y	Y	Y	Y

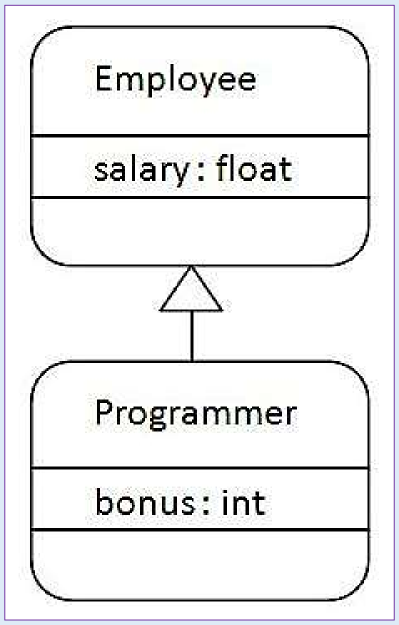
❖The syntax of Java Inheritance



```
class Subclass-name extends Superclass-name
{
   //methods and fields
}
```

- ➢The extends keyword indicates that you are making a new class that derives from an existing class.
- ➢The meaning of "extends" is to increase the functionality.
- ➢In the terminology of Java, a class which is inherited is called a parent or superclass, and the new class is called child or subclass.

- Programmer is the subclass (child class)
- Employee is the superclass (Parent class)
- The relationship between the two classes is Programmer IS-A Employee
- It means that Programmer is a type of Employee.



```
class Employee{
 float salary=40000;
}
class Programmer extends Employee{
 int bonus=10000;
 public static void main(String args[]){
   Programmer p=new Programmer();
   System.out.println("Programmer salary is:"+p.salary);
   System.out.println("Bonus of Programmer is:"+p.bonus);
}
}
```

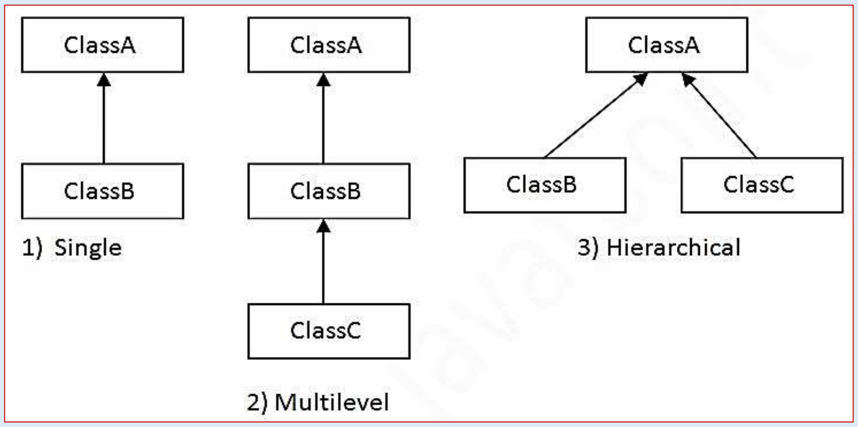
Output

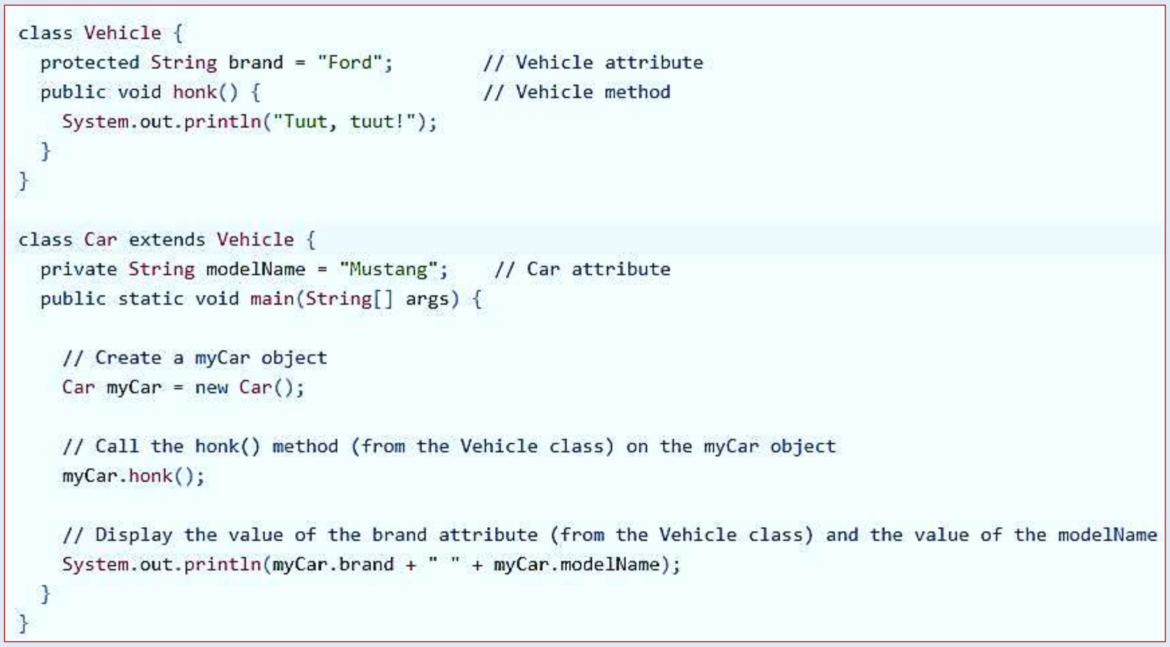
```
Programmer salary is:40000.0
Bonus of programmer is:10000
```

- Programmer object can access the attribute of its own class as well as of Employee class i.e. code reusability.

❖Types of inheritance in java

➢On the basis of class, there can be three types of inheritance in java: single, multilevel and hierarchical.





```
class Vehicle {
  protected String brand = "Ford";        // Vehicle attribute
  public void honk() {                    // Vehicle method
    System.out.println("Tuut, tuut!");
  }
}

class Car extends Vehicle {
  private String modelName = "Mustang";    // Car attribute
  public static void main(String[] args) {

    // Create a myCar object
    Car myCar = new Car();

    // Call the honk() method (from the Vehicle class) on the myCar object
    myCar.honk();

    // Display the value of the brand attribute (from the Vehicle class) and the value of the modelName
    System.out.println(myCar.brand + " " + myCar.modelName);
  }
}
```

SUPER KEYWORD

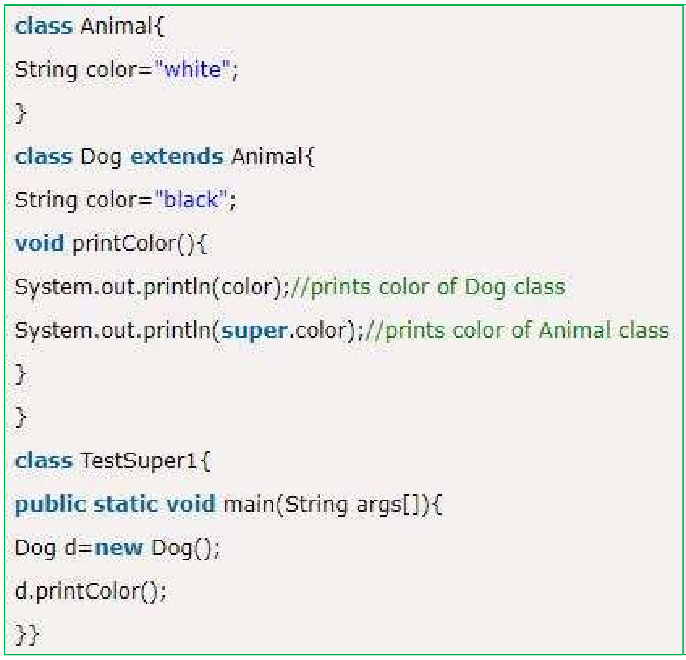
- The super keyword in Java is a reference variable which is used to refer immediate parent class object.

Usage of Java super Keyword

- super can be used to refer immediate parent class instance variable.
- super can be used to invoke immediate parent class method.
- super() can be used to invoke immediate parent class constructor.

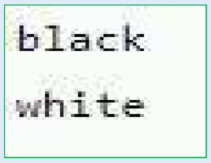
- We can use super keyword to access the data member (attribute) of parent class. It is used if parent class and child class have same attribute.

Prepared By Mr. EBIN PM, AP, IESCE EDULINE 1



```
class Animal{
String color="white";
}
class Dog extends Animal{
String color="black";
void printColor(){
System.out.println(color);//prints color of Dog class
System.out.println(super.color);//prints color of Animal class
}
}
class TestSuper1{
public static void main(String args[]){
Dog d=new Dog();
d.printColor();
}}
```

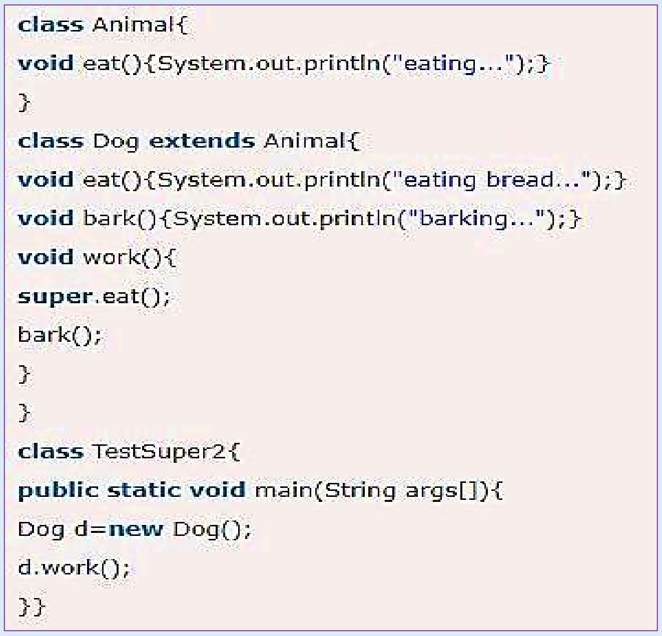
output



```
black
white
```

Animal and Dog both classes have a common property color. If we print color property, it will print the color of current class by default. To access the parent property, we need to use super keyword.

Prepared By Mr. EBIN PM, AP, IESCE EDULINE 2



```
class Animal{
void eat(){System.out.println("eating...");}
}
class Dog extends Animal{
void eat(){System.out.println("eating bread...");}
void bark(){System.out.println("barking...");}
void work(){
super.eat();
bark();
}
}
class TestSuper2{
public static void main(String args[]){
Dog d=new Dog();
d.work();
}}
```

Output

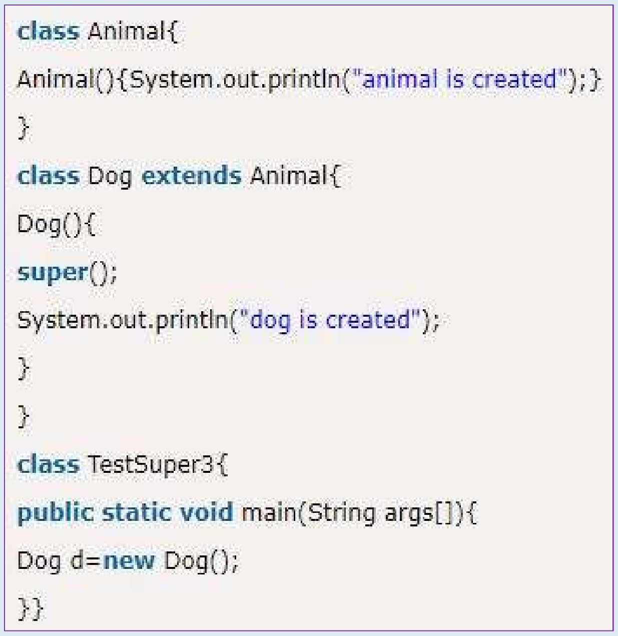
```
eating...
barking...
```

The super keyword can also be used to invoke(call) parent class method.

Prepared By Mr. EBIN PM, AP, IESCE EDULINE 13

- In the above example Animal and Dog both classes have eat() method
- If we call eat() method from Dog class, it will call the eat() method of Dog class by default because priority is given to local.
- To call the parent class method, we need to use super keyword.

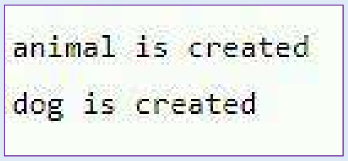
Prepared By Mr. EBIN PM, AP, IESCE EDULINE 14



```
class Animal{
Animal(){System.out.println("animal is created");}
}
class Dog extends Animal{
Dog(){
super();
System.out.println("dog is created");
}
}
class TestSuper3{
public static void main(String args[]){
Dog d=new Dog();
}}
```

The super keyword can also be used to invoke the parent class constructor.

Output



```
animal is created
dog is created
```

Calling order of constructors in inheritance

- Order of execution of constructors in inheritance relationship is from base (parent) class to derived (child)class.
- We know that when we create an object of a class then the constructors get called automatically.
- In inheritance relationship, when we create an object of a child class, then first base class constructor and then derived class constructor get called implicitly.
- In simple word, we can say that the parent class constructor get called first, then of the child class constructor.

```
class A {
  A() {
        System.out.println("Inside A's constructor.");
       }
 }
// Create a subclass by extending class A.
class B extends A {
  B() {
        System.out.println("Inside B's constructor.");
       }
}
// Create another subclass by extending B.
class C extends B {
  C() {
        System.out.println("Inside C's constructor.");
       }
}
public class Main{
    public static void main(String args[])
      {
        C c = new C();
      }
}
```

Output

```
Inside A's constructor.
Inside B's constructor.
Inside C's constructor.
```

METHOD OVERRIDING

- If subclass (child class) has the same method as declared in the parent class, it is known as method overriding in Java.
- In other words, If a subclass provides the specific implementation of the method that has been declared by one of its parent class, it is known as method overriding

Usage of Java Method Overriding

- Method overriding is used to provide the specific implementation of a method which is already provided by its superclass.
- Method overriding is used for runtime polymorphism

Rules for Java Method Overriding

- The method must have the same name as in the parent class
- The method must have the same parameter as in the parent class.
- There must be an IS-A relationship (inheritance).

Remember…….

- A static method cannot be overridden. It is because the static method is bound with class whereas instance method is bound with an object. Static belongs to the class area, and an instance belongs to the heap area.
- Can we override java main method? - No, because the main is a static method.

Example - method overriding

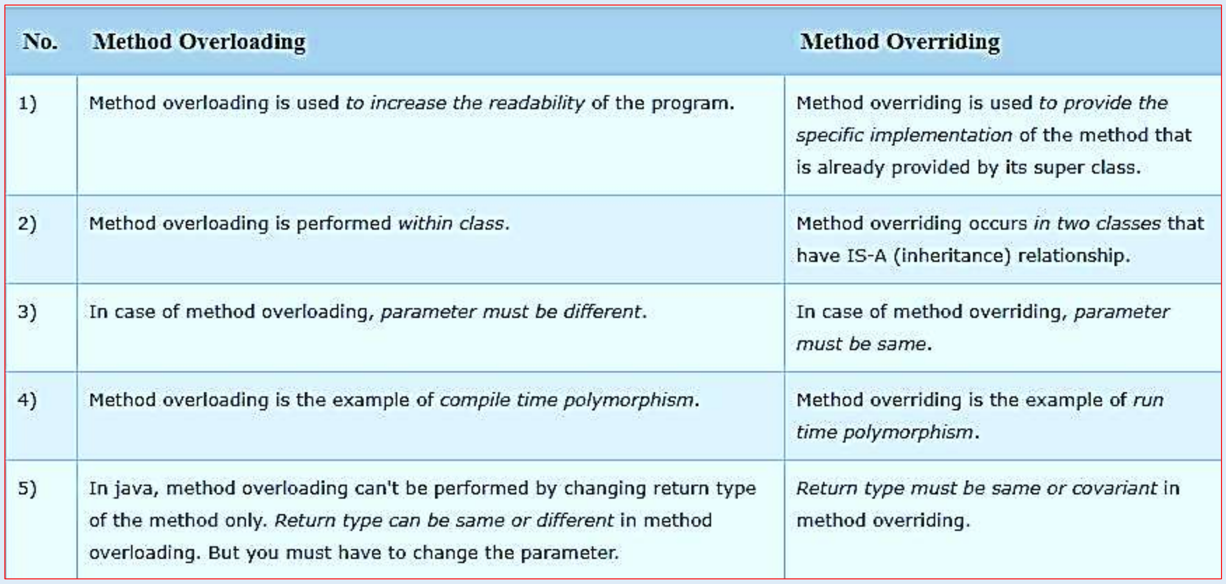
```
//Java Program to illustrate the use of Java Method Overriding
//Creating a parent class.
class Vehicle{
  //defining a method
  void run(){System.out.println("Vehicle is running");}
}
//Creating a child class
class Bike2 extends Vehicle{
  //defining the same method as in the parent class
  void run(){System.out.println("Bike is running safely");}

  public static void main(String args[]){
  Bike2 obj = new Bike2();//creating object
  obj.run();//calling method
  }
}
```

Output

```
Bike is running safely
```

we have defined the run method in the subclass as defined in the parent class but it has some specific implementation. The name and parameter of the method are the same, and there is IS-A relationship between the classes, so there is method overriding.

method overloading Vs. method overriding



No.	Method Overloading	Method Overriding
1)	Method overloading is used *to increase the readability* of the program.	Method overriding is used *to provide the specific implementation* of the method that is already provided by its super class.
2)	Method overloading is performed *within class.*	Method overriding occurs *in two classes* that have IS-A (inheritance) relationship.
3)	In case of method overloading, *parameter must be different*.	In case of method overriding, *parameter must be same*.
4)	Method overloading is the example of *compile time polymorphism*.	Method overriding is the example of *run time polymorphism*.
5)	In java, method overloading can't be performed by changing return type of the method only. *Return type can be same or different* in method overloading. But you must have to change the parameter.	*Return type must be same or covariant* in method overriding.

Prepared By Mr. EBIN PM, AP, IESCE

FINAL KEYWORD

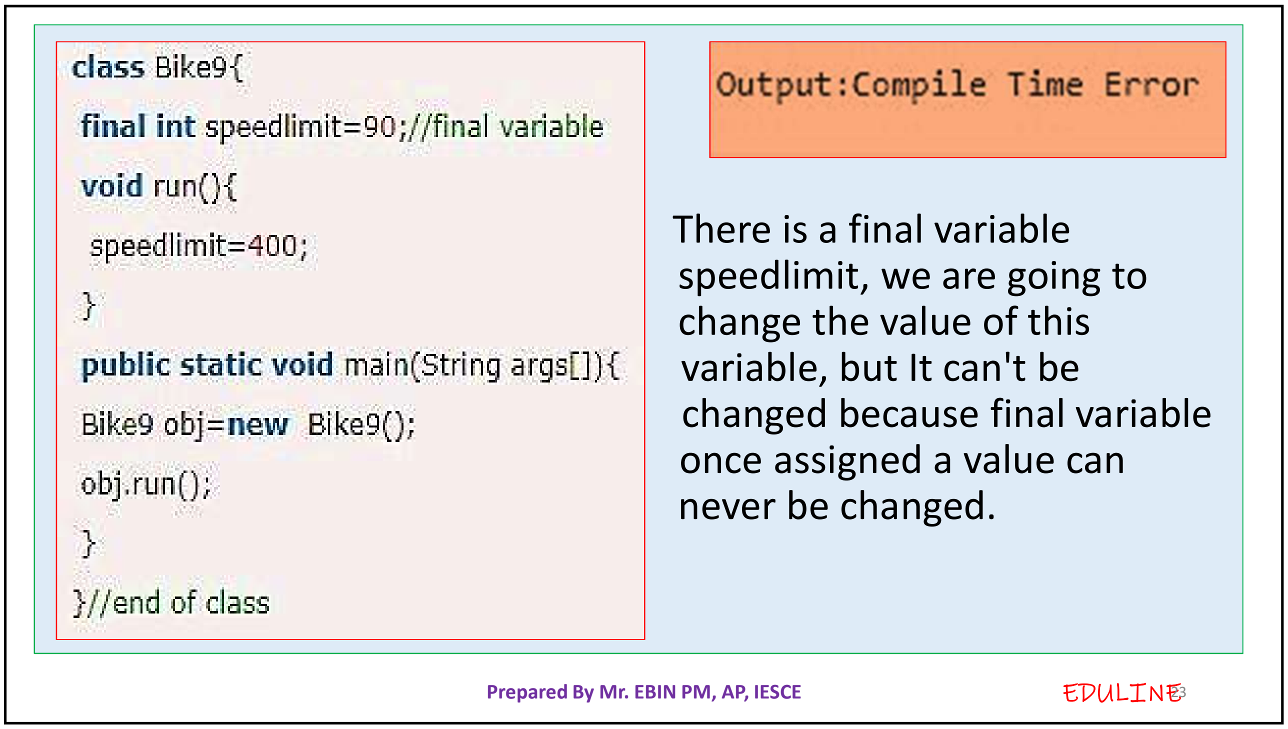
➢ The final keyword in java is used to restrict the user. The java final keyword can be used in many context. Final can be:

1. variable
2. method
3. class

❖ **Java final variable**

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

Prepared By Mr. EBIN PM, AP, IESCE



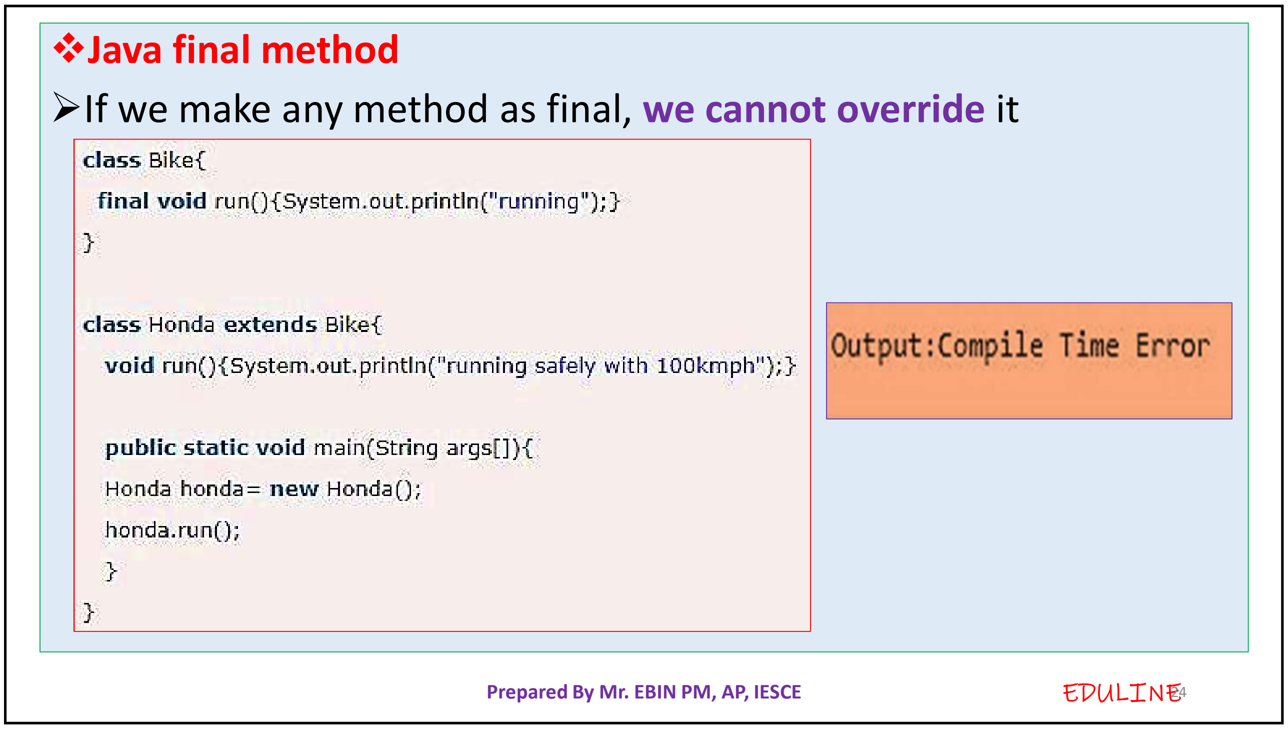
```
class Bike9{
 final int speedlimit=90;//final variable
 void run(){
  speedlimit=400;
 }
 public static void main(String args[]){
 Bike9 obj=new  Bike9();
 obj.run();
 }
}//end of class
```

```
Output:Compile Time Error
```

There is a final variable speedlimit, we are going to change the value of this variable, but It can't be changed because final variable once assigned a value can never be changed.

Prepared By Mr. EBIN PM, AP, IESCE

EDULINE



❖Java final method

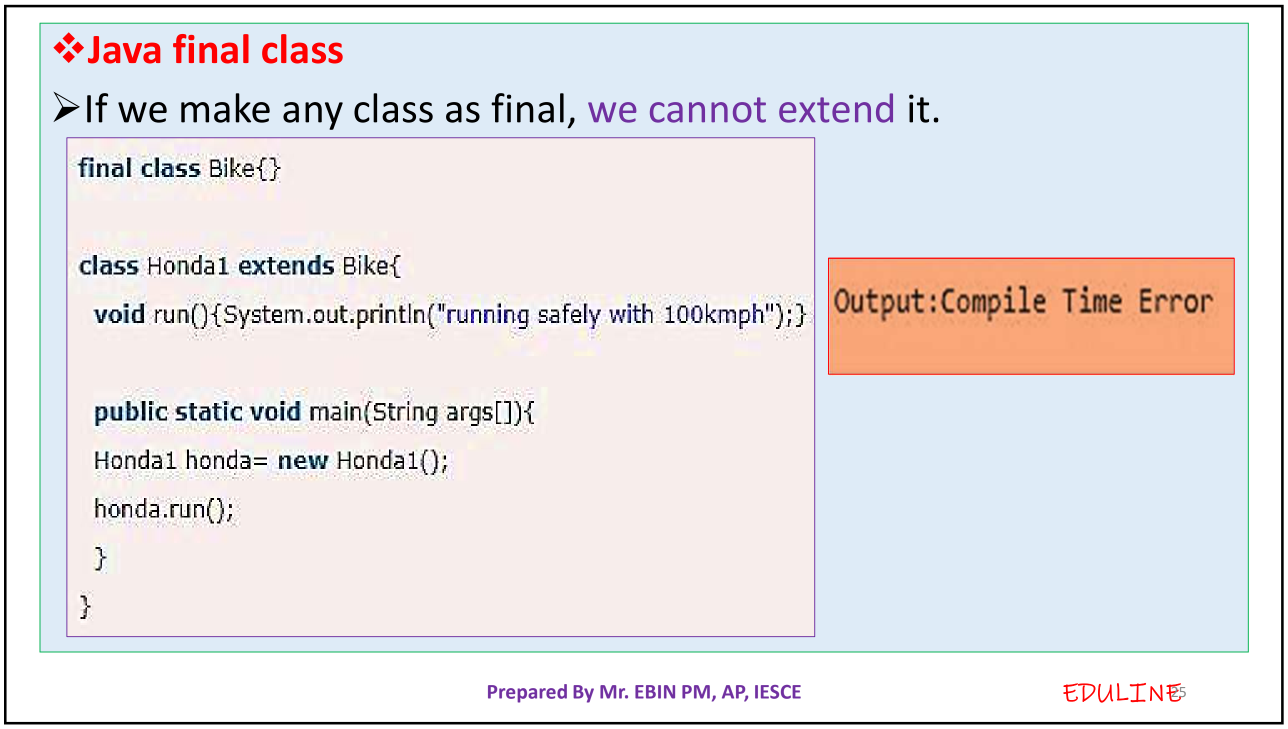
➢If we make any method as final, **we cannot override** it

```
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
```

Prepared By Mr. EBIN PM, AP, IESCE

EDULINE



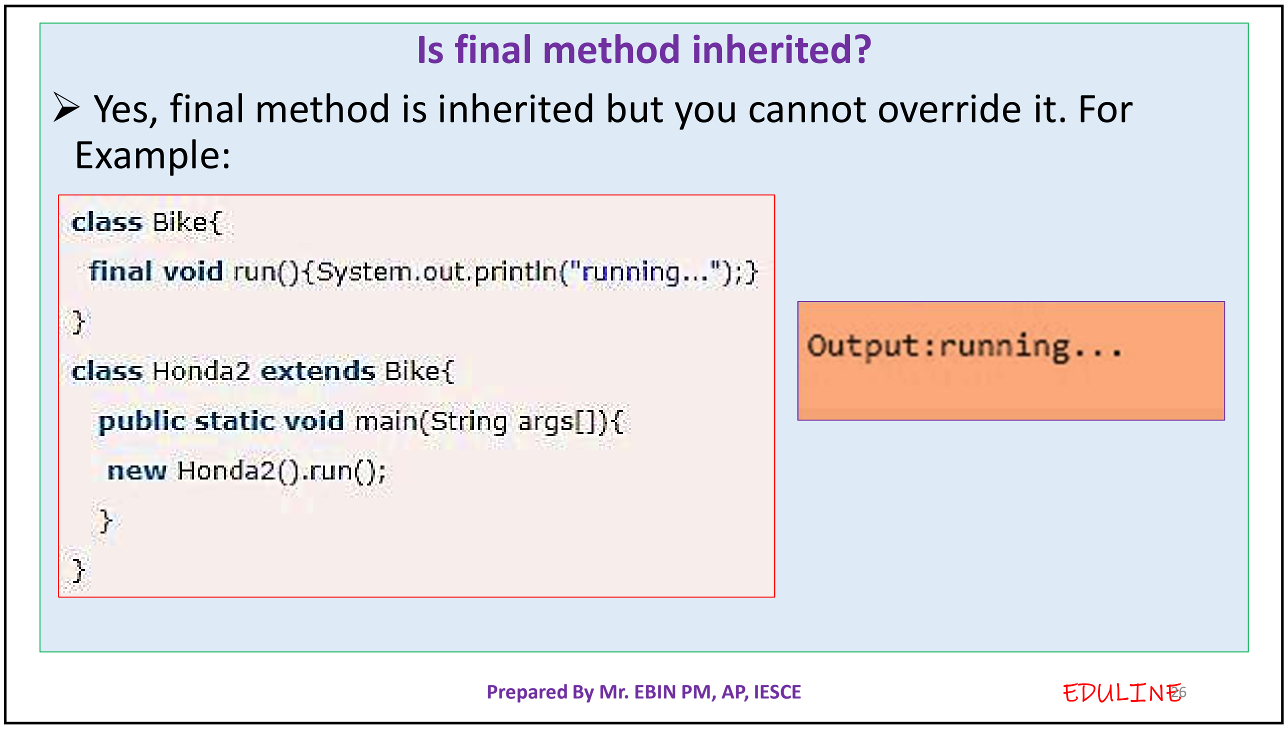
❖Java final class

➢If we make any class as final, we cannot extend it.

```
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 Honda1();
  honda.run();
  }
}
```

```
Output:Compile Time Error
```

Prepared By Mr. EBIN PM, AP, IESCE 25



Is final method inherited?

➢ Yes, final method is inherited but you cannot override it. For Example:

```
class Bike{
  final void run(){System.out.println("running...");}
}
class Honda2 extends Bike{
   public static void main(String args[]){
    new Honda2().run();
   }
}
```

```
Output:running...
```

Prepared By Mr. EBIN PM, AP, IESCE 26

❖ **Points to Remember**

1) A constructor cannot be declared as final.

2) Local final variable must be initializing during declaration.

3) We cannot change the value of a final variable.

4) A final method cannot be overridden.

5) A final class not be inherited.

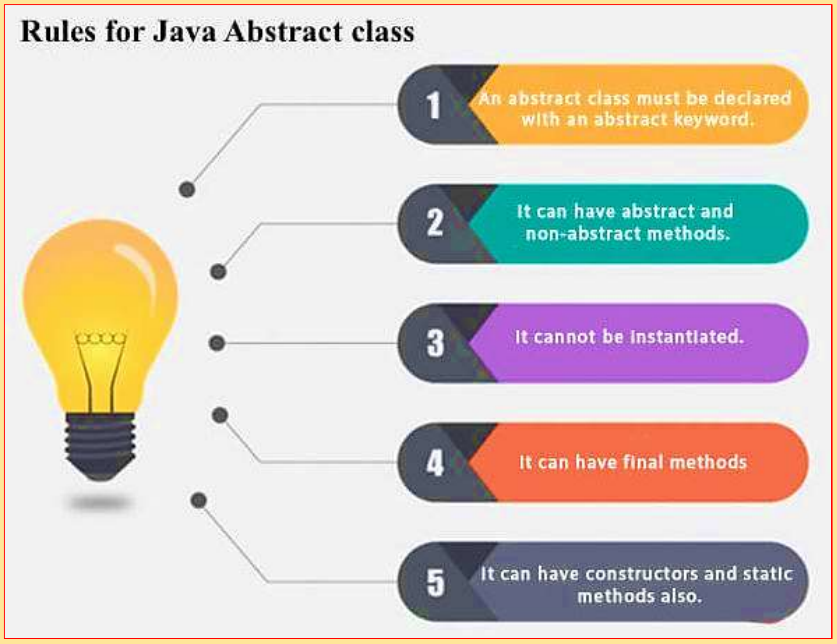
6) If method parameters are declared final then the value of these parameters cannot be changed.

7) final, finally and finalize are three different terms. finally is used in exception handling and finalize is a method that is called by JVM during garbage collection.

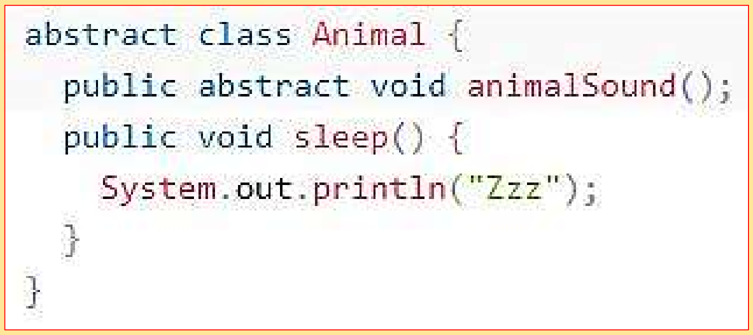
ABSTRACT CLASSES AND METHODS

- Data abstraction is the process of hiding certain details and showing only essential information to the user.
- **Abstract class:** is a restricted class that cannot be used to create objects (to access it, it must be inherited from another class).
- **Abstract method:** can only be used in an abstract class, and it does not have a body. The body is provided by the subclass (inherited from).
- An abstract class can have both abstract and regular methods:

❖Abstract class


Rules for Java Abstract class

1. An abstract class must be declared with an abstract keyword.
2. It can have abstract and non-abstract methods.
3. It cannot be instantiated.
4. It can have final methods
5. It can have constructors and static methods also.

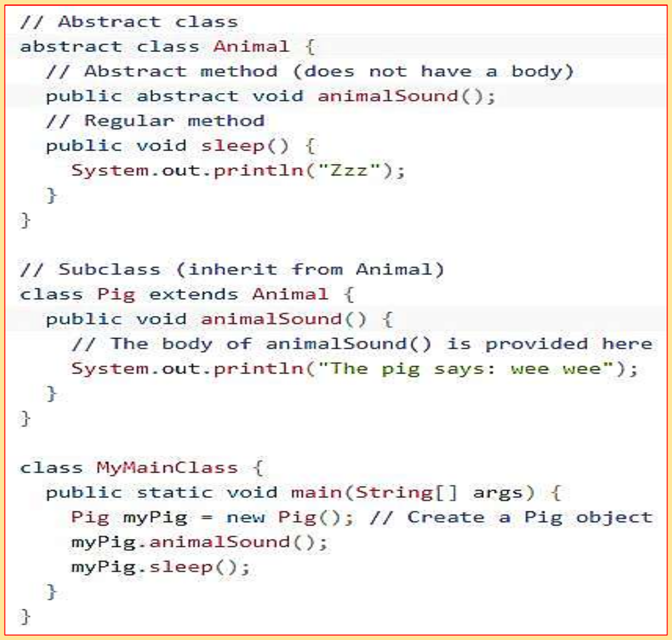

```
abstract class Animal {
  public abstract void animalSound();
  public void sleep() {
    System.out.println("Zzz");
  }
}
```

➢From the example above, it is not possible to create an object of the Animal class

Animal myObj = new Animal(); // will generate an error

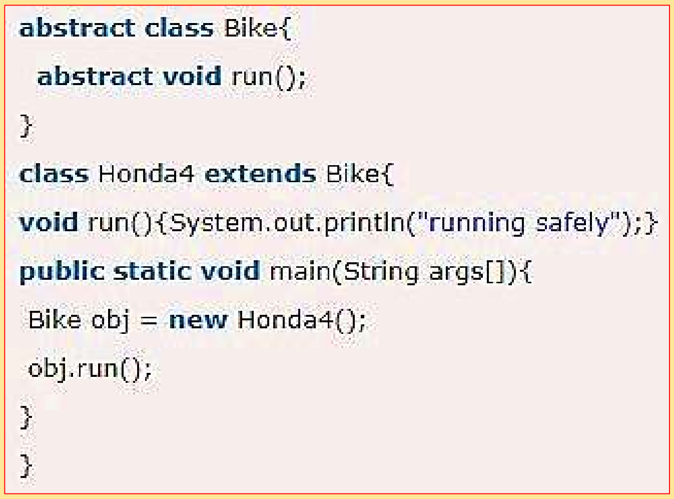
➢To access the abstract class, it must be inherited from another class

Example



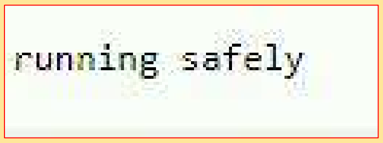
```
// Abstract class
abstract class Animal {
  // Abstract method (does not have a body)
  public abstract void animalSound();
  // Regular method
  public void sleep() {
    System.out.println("Zzz");
  }
}

// Subclass (inherit from Animal)
class Pig extends Animal {
  public void animalSound() {
    // The body of animalSound() is provided here
    System.out.println("The pig says: wee wee");
  }
}

class MyMainClass {
  public static void main(String[] args) {
    Pig myPig = new Pig(); // Create a Pig object
    myPig.animalSound();
    myPig.sleep();
  }
}
```

Example - Here Bike is an abstract class that contains only one abstract method run. Its implementation is provided by the Honda class.



```
abstract class Bike{
  abstract void run();
}
class Honda4 extends Bike{
void run(){System.out.println("running safely");}
public static void main(String args[]){
 Bike obj = new Honda4();
 obj.run();
}
}
```

Output

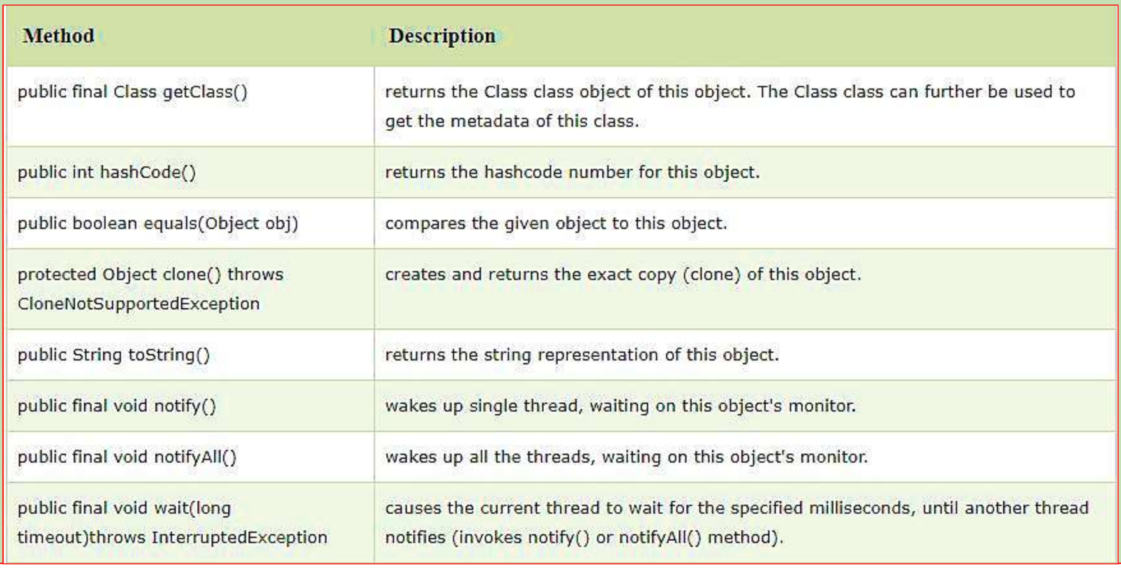


```
running safely
```

THE OBJECT CLASS

- The Object class is the parent class of all the classes in java by default. In other words, it is the topmost class of java.
- The Object class provides some common behaviors to all the objects such as object can be compared, object can be cloned, object can be notified etc.
- Object class is present in java.lang package
- Every class in Java is directly or indirectly derived from the Object class

❖Methods of Object class



Method	Description
public final Class getClass()	returns the Class class object of this object. The Class class can further be used to get the metadata of this class.
public int hashCode()	returns the hashcode number for this object.
public boolean equals(Object obj)	compares the given object to this object.
protected Object clone() throws CloneNotSupportedException	creates and returns the exact copy (clone) of this object.
public String toString()	returns the string representation of this object.
public final void notify()	wakes up single thread, waiting on this object's monitor.
public final void notifyAll()	wakes up all the threads, waiting on this object's monitor.
public final void wait(long timeout)throws InterruptedException	causes the current thread to wait for the specified milliseconds, until another thread notifies (invokes notify() or notifyAll() method).