OOP: Fundamentals

Agenda:

Inheritance : Multiple & Multilevel
Polymorphism: Overloading & Overriding
Execution Sequence of Constructors, DMD-Dynamic Method Dispatch, Super and Final.

Md. Mamun Hossain
B.Sc. (Engg.) & M.Sc. (Thesis) in CSE , SUST
Assistant Professor, Dept. of CSE, BAUST

Java doesn't support multiple inheritance through class. Java Static Properties is shared to all objects

OOP Fundamentals: Core Concepts

Keywords:

- ✓ **Class, Object: Declaration and Assignment**
- ✓ **Method: Simple& Parameterized,**
- ✓ **Constructor: Default& Parameterized**
- ✓ **The dot(.) operator, initialization of variable: Three ways**
- ✓ ***this* keywords, Garbage Collection, Finalize method()**
- ✓ **Access Modifiers,**
- ✓ **Java Static Properties: Variables and Methods**
- ✓ **Var-arg and Command line arg.**
- ✓ **Inheritance: Multiple and Multilevel**
- ✓ **Polymorphism: Overloading & Overriding**
- ✓ **Execution Sequence of Constructors,**
- ✓ **DMD-Dynamic Method Dispatch, Super and Final.**

Inheritance : Basic Concept

- Inheritance in Java is a mechanism in which one object (child) acquires all the properties and behaviors of another (parent) object. It is on of the most important principal of OOPs.

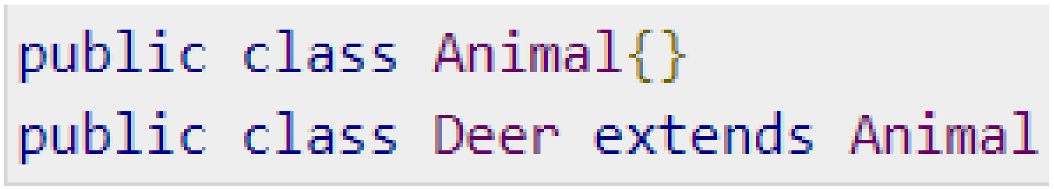
- The idea behind inheritance in Java is that you can create new classes that are built upon existing classes. When you inherit from an existing class, you can reuse methods and fields of the parent class. Moreover, you can add new methods and fields in your current class also.

- Inheritance represents the ***IS-A relationship*** which is also known as ***a parent-child relationship***.

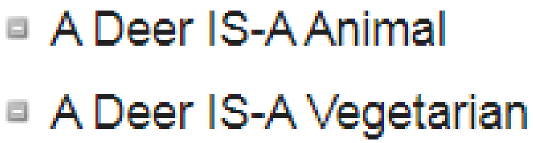
Inheritance: IS-A Relationship

- Any Java object that can pass more than one IS-A (which is also known as a *parent-child* relationship)test is considered to be **IS-A** relationship.

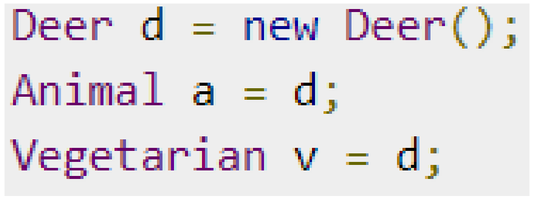
Example : Let us look at an example.



```
public class Animal{}
public class Deer extends Animal
```



- A Deer IS-A Animal
- A Deer IS-A Vegetarian



```
Deer d = new Deer();
Animal a = d;
Vegetarian v = d;
```

Md. Mamun Hossain
B. Sc. (Engg.) & M. Sc. (Thesis) in CSE, SUST
Asst. Professor, Dept. of CSE , BAUST

Inheritance: Has-A Relationship

- **Aggregation**

  If a class have an entity reference, it is known as Aggregation. Aggregation represents HAS-A relationship.

- **Example** : Let us look at an example.

  Consider a situation, Employee object contains many information such as id, name, email ID etc. It contains one more object named address, which contains its own information's such as city, state, country, zip code etc. as given below.

```
class Employee{
      int id;
      String name;
      Address address;
     //Address is a class
      … … …
      … … …
      … … …
}
```

```
public class Address {
         String city, state, country;
        public Address(String ct, String st, String cy)
         {
                    city = ct;
                    state = st;
                    country = cy;
        }
 }
```

In such case, Employee has an entity reference address, so relationship is Employee HAS-A address.

Md. Mamun Hossain
B. Sc. (Engg.) & M. Sc. (Thesis) in CSE, SUST
Asst. Professor, Dept. of CSE , BAUST

Why and When Aggregation?

- **Why use Aggregation?**

  For Code Reusability.

- **When use Aggregation?**

  Code reuse is also best achieved by aggregation when there is no is-a relationship. Inheritance should be used only if the relationship is-a is maintained throughout the lifetime of the objects involved; otherwise, aggregation is the best choice.

Inheritance : Terminologies

- **Class**: A class defines the common properties of a group of objects. It is a ***template or blueprint*** from which objects are created.
- **Sub Class:** Subclass is a class which inherits the other class. It is also called a ***derived class, extended class, or child class***.
- **Super Class:** Superclass is the class from where a subclass inherits the features. It is also called a ***base class or parent class***.

Properties:

- **Reusability:** As the name specifies, reusability is a mechanism which facilitates you to reuse the fields(variables) and methods of the existing class (parent) when you create a new (child) class. You can use the same fields and methods already defined in the previous class.

Md. Mamun Hossain
B. Sc. (Engg.) & M. Sc. (Thesis) in CSE, SUST
Asst. Professor, Dept. of CSE , BAUST

Inheritance: At a glance

Java Inheritance (Subclass and Superclass)

In Java, it is possible to inherit attributes and methods from one class to another.

We group the "inheritance concept" into two categories:

- **subclass** (child)- the class that inherits from another class
- **superclass** (parent)- the class being inherited from

*To inherit from a class, use the **extends** keyword.*

Md. Mamun Hossain
B. Sc. (Engg.) & M. Sc. (Thesis) in CSE, SUST
Asst. Professor, Dept. of CSE , BAUST

Inheritance : Category

There are two type of Inheritance in Java

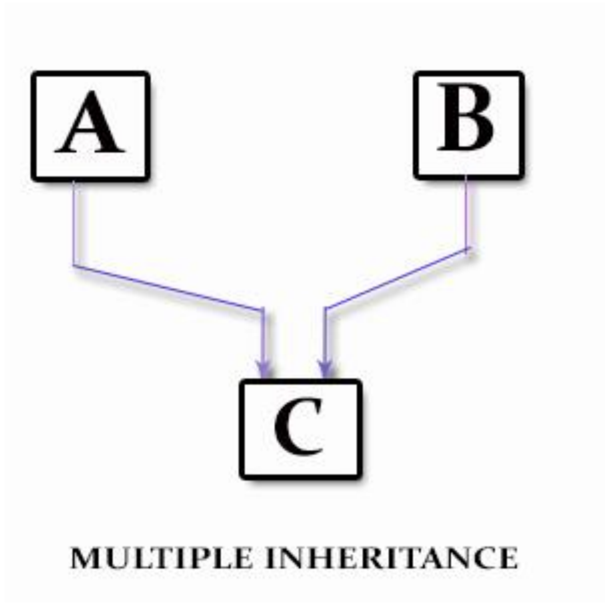
1. **Multiple** – only achieve through Interface
2. **Multilevel** – can be achieved through both class and Interface

NB: *If you are familiar with C++ where both multiple (diamond problem and virtual keyword) and multilevel inheritance can be achieved through class but careful this is not the case in Java.*

Md. Mamun Hossain
B. Sc. (Engg.) & M. Sc. (Thesis) in CSE, SUST
Asst. Professor, Dept. of CSE , BAUST

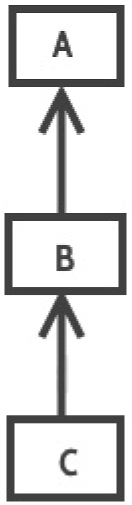
Inheritance : Multiple & Multilevel

Multiple Inheritance is an **Inheritance** type where a class **inherits** from more than one base class. In Java, *it can only achieved through Interface.*



MULTIPLE INHERITANCE

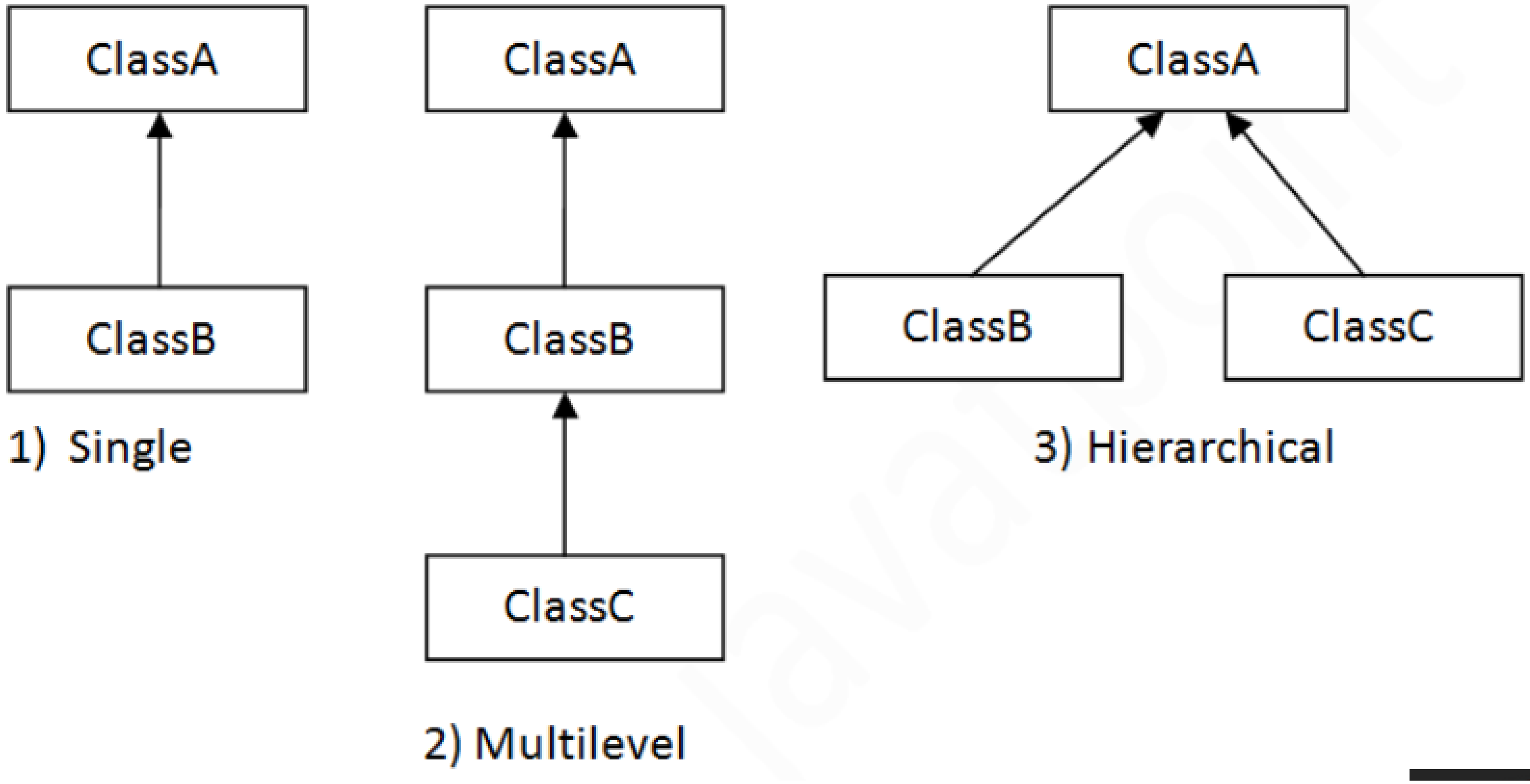
Multilevel Inheritance is an **Inheritance** type that **inherits** from a derived class, making that derived class a base class for a new class.



Multilevel Inheritance

Multilevel Inheritance : Type

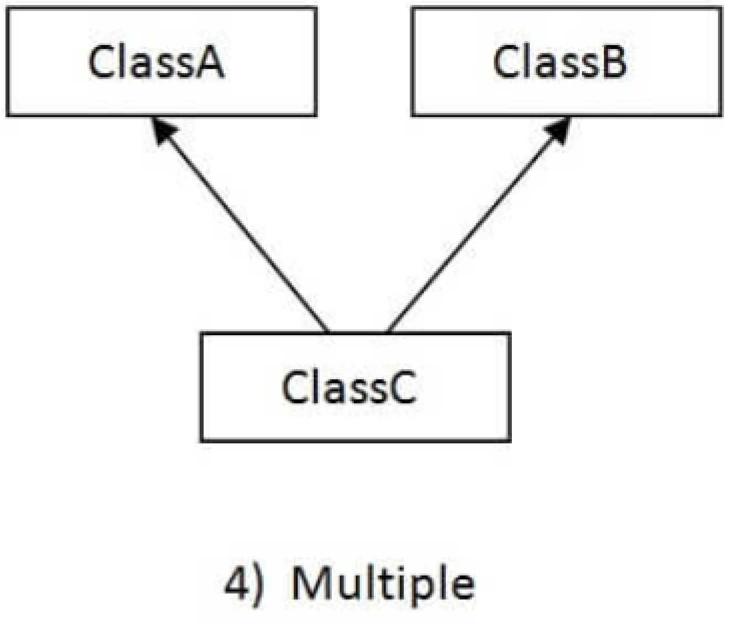
- On the basis of class, there can be three types of inheritance in java: single, multilevel and hierarchical.
- In java programming, multiple and hybrid inheritance is supported through interface only. We will learn about interfaces later.



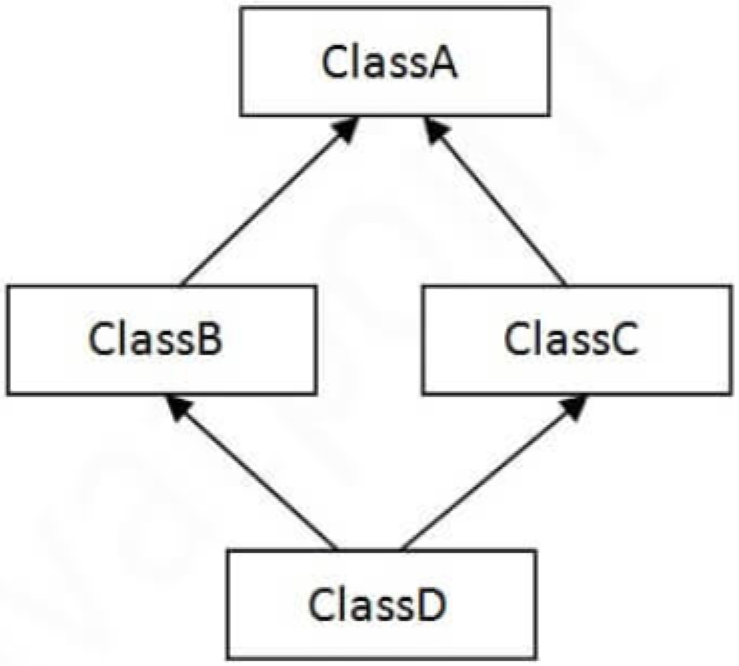
Multiple Inheritance in java

Note: Multiple inheritance is not supported in Java through class.

When one class inherits multiple classes, it is known as multiple inheritance. For Example:



4) Multiple



5) Hybrid

We can achieve multiple inheritance in java only through Interface

Why multiple inheritance is not supported through class in java?

- To reduce the complexity and simplify the language, multiple inheritance is not supported in java.
- Consider a scenario where A, B, and C are three classes. The C class inherits A and B classes. If A and B classes have the same method and you call it from child class object, there will be ambiguity to call the method of A or B class.
- Since compile-time errors are better than runtime errors, Java renders compile-time error if you inherit 2 classes.

So whether you have same method or different, there will be compile time error.

```
class A{
void msg(){System.out.println("Hello");}
}
class B{
void msg(){System.out.println("Welcome");}
}
class C extends A,B{//suppose if it were

 public static void main(String args[]){
  C obj=new C();
  obj.msg();//Now which msg() method would be invoked?
}
}
```

Output

Compile time Error

Md. Mamun Hossain
B. Sc. (Engg.) & M. Sc. (Thesis) in CSE, SUST
Asst. Professor, Dept. of CSE , BAUST

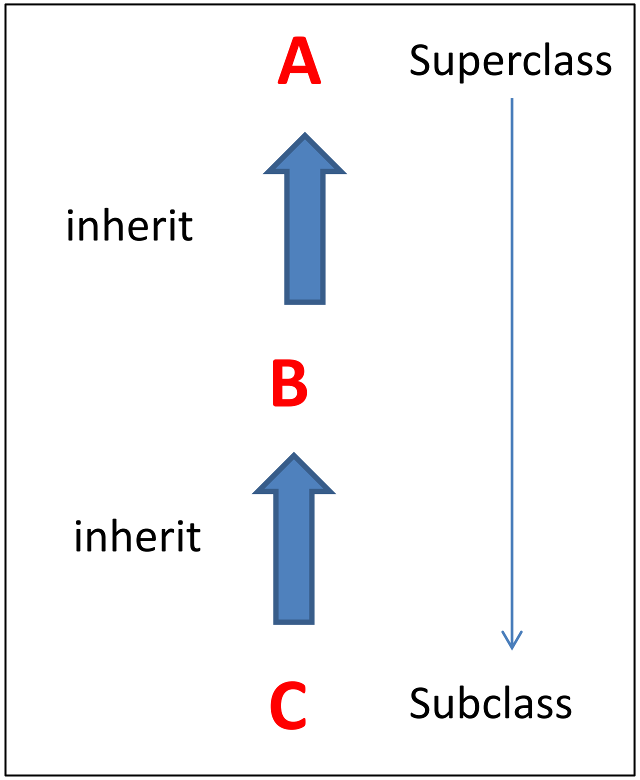
Inheritance (Multilevel) : Example

In the example below, class C (subclass) inherits the attributes and methods from class B and the B class (subclass) inherits the attributes and methods (Not Constructors) from the A class (superclass).

```
class A{
        int  a;
       void show(){  …  }
}
```

```
class B extends A{
         int b;
         void msg(){ …}
}
```

```
class C extends B{
         int c;
         void hello(){ …}
}
```



Md. Mamun Hossain
B. Sc. (Engg.) & M. Sc. (Thesis) in CSE, SUST
Asst. Professor, Dept. of CSE , BAUST

Inheritance (Multilevel) : Example

In the example below, the B class (subclass) inherits the attributes and methods (Not Constructors) from the A class (superclass).

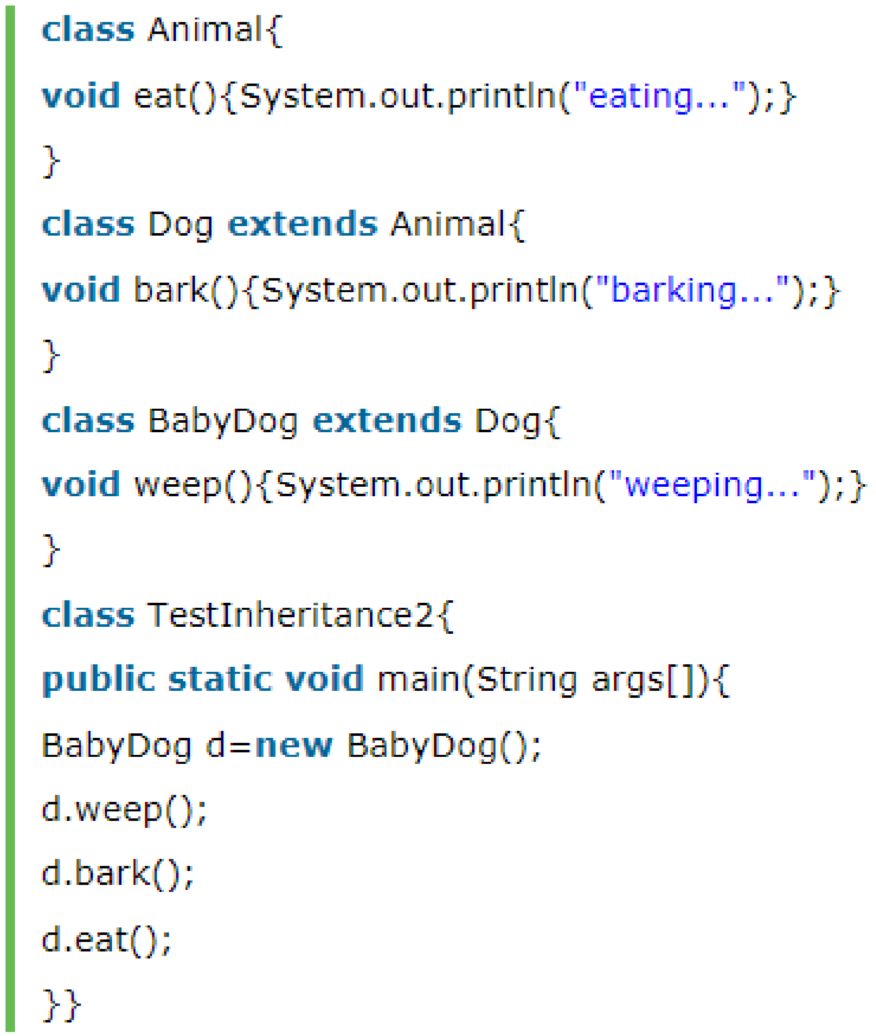
```
class A{
        int  a;
        A(){ … }
       void show1(){  …  }
 }
```

```
class B extends A{
         int b;
         B(){ … }
         void show2(){ …}
 }
```

```
public class Access {
   public static void main(String args[]) {
            A  ob1=new A();
               ob1.a=10
               ob1.show1();

            B ob2=new B();
              ob2.b=30;
               ob2. show2();

              ob2.a=20;
               ob2.show1();
             }
}
```

Md. Mamun Hossain
B. Sc. (Engg.) & M. Sc. (Thesis) in CSE, SUST
Asst. Professor, Dept. of CSE , BAUST

Inheritance (Multilevel) : Example



```
class Animal{
void eat(){System.out.println("eating...");}
}
class Dog extends Animal{
void bark(){System.out.println("barking...");}
}
class BabyDog extends Dog{
void weep(){System.out.println("weeping...");}
}
class TestInheritance2{
public static void main(String args[]){
BabyDog d=new BabyDog();
d.weep();
d.bark();
d.eat();
}}
```

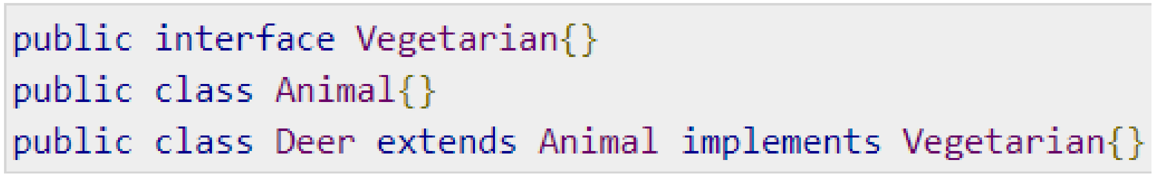
Polymorphism : Background

- Polymorphism means "many forms", and it occurs when we have many classes that are related to each other by inheritance.
- Polymorphism is the ability of an object to take on many forms.

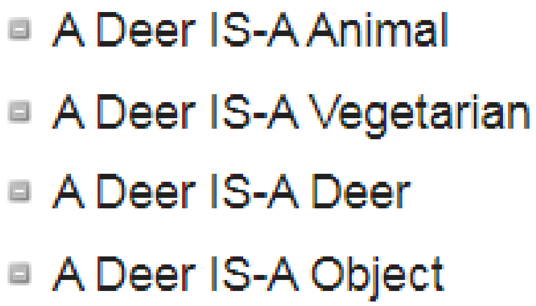
o **For example**, think of a superclass called Animal that has a method called animal Sound(). Subclasses of Animals could be Pigs, Cats, Dogs, Birds - And they also have their own implementation of an animal sound (the pig oinks, and the cat meows, etc.)

Polymorphism : IS-A property

- Any Java object that can pass more than one IS-A (which is also known as a *parent-child* relationship) test is considered to be **polymorphic**.

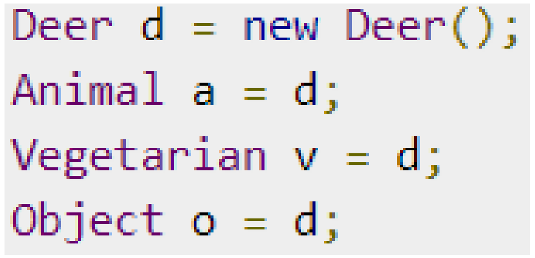


```
public interface Vegetarian{}
public class Animal{}
public class Deer extends Animal implements Vegetarian{}
```



- A Deer IS-A Animal
- A Deer IS-A Vegetarian
- A Deer IS-A Deer
- A Deer IS-A Object

The most common use of polymorphism in OOP occurs when a parent class reference is used to refer to a child class object.



```
Deer d = new Deer();
Animal a = d;
Vegetarian v = d;
Object o = d;
```

In Java, all Java objects are polymorphic since any object will pass the IS-A test for their own type and for the class Object.

Md. Mamun Hossain
B. Sc. (Engg.) & M. Sc. (Thesis) in CSE, SUST
Asst. Professor, Dept. of CSE , BAUST

Polymorphism : Category

Polymorphism can be categorized into:

- **Overloading:**
  - Different methods have the same name, but different signatures
  - Overloading is related to compile-time (or static) polymorphism.
  - Also referred as early binding
- **Overriding**
  - Different methods (of superclass and subclass) have the same name and same signatures but different body implementation
  - Overloading is related to run-time (or dynamic) polymorphism.
  - Also referred as late binding

Return type is not sufficient to distinguish between two method.

Md. Mamun Hossain
B. Sc. (Engg.) & M. Sc. (Thesis) in CSE, SUST
Asst. Professor, Dept. of CSE , BAUST

Polymorphism : Overloading

- If a class has multiple methods having same name but different in parameters, it is known as **Method Overloading**.

- Advantage of method overloading

  Method overloading *increases the readability of the program*.

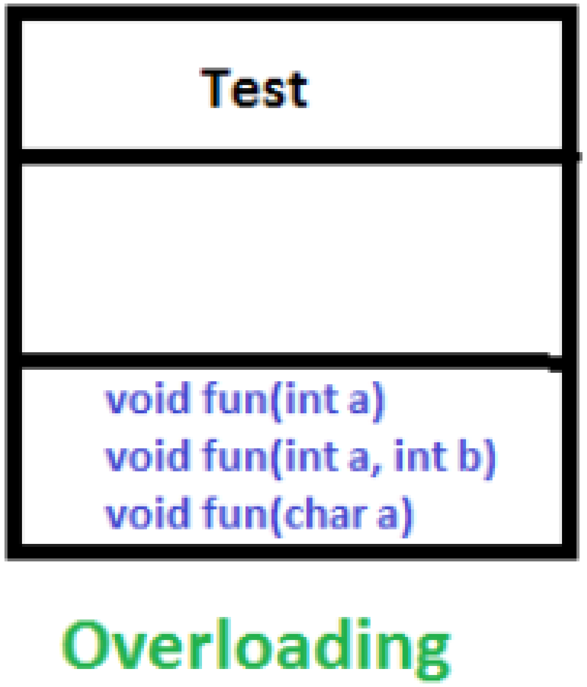
- Different ways to overload the method
  - ✓ There are two ways to overload the method in java
  - ✓ By changing number of arguments
  - ✓ By changing the data type

In java, Method Overloading is not possible by changing the return type of the method only.

Md. Mamun Hossain
B. Sc. (Engg.) & M. Sc. (Thesis) in CSE, SUST
Asst. Professor, Dept. of CSE , BAUST

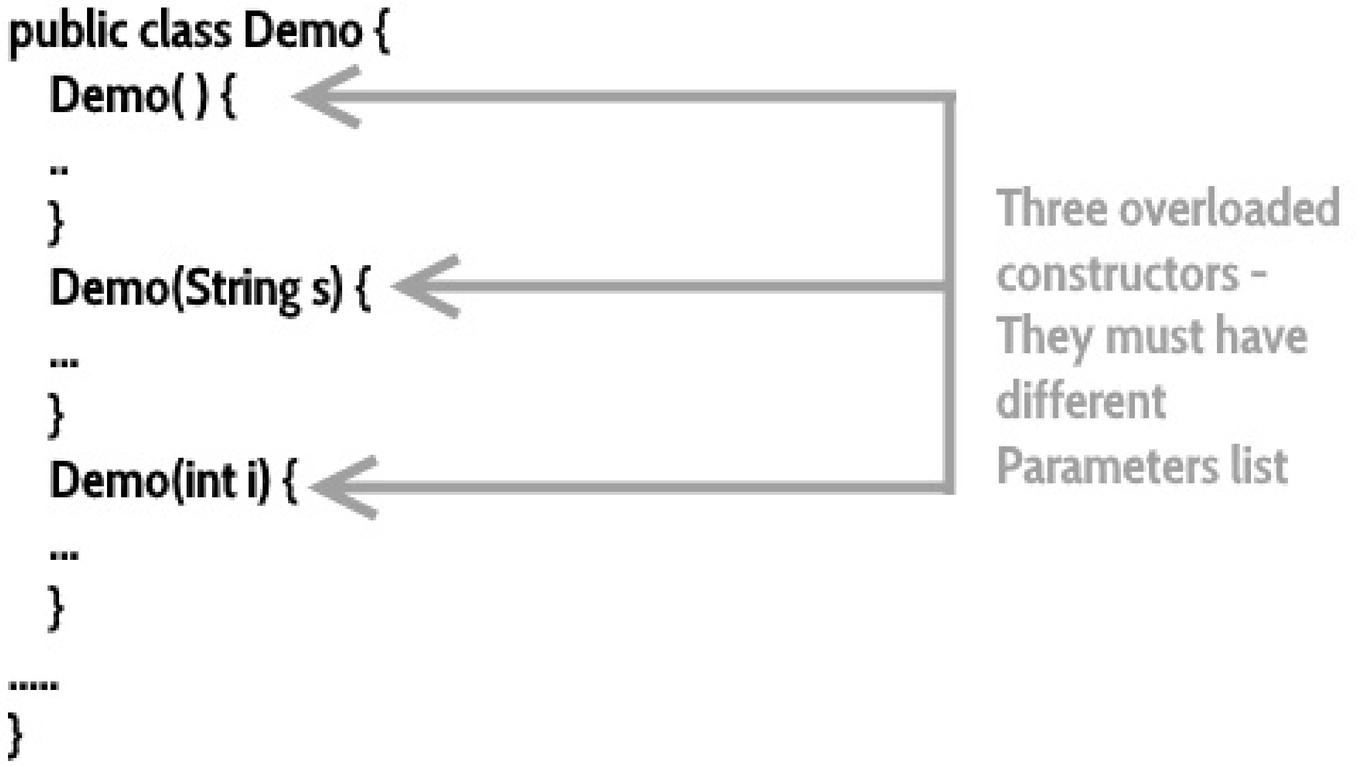
Polymorphism : Overloading

- **Overloading**
  - Overloading allows different methods to have the same name, but different signatures where the signature can differ by the number of input parameters or type of input parameters or both.
  - Overloading is related to compile-time (or static) polymorphism.



Md. Mamun Hossain
B. Sc. (Engg.) & M. Sc. (Thesis) in CSE, SUST
Asst. Professor, Dept. of CSE , BAUST

Polymorphism : Constructor Overloading



```
public class Demo {
   Demo( ) {
   ..
   }
   Demo(String s) {
   ...
   }
   Demo(int i) {
   ...
   }
.....
}
```

Three overloaded constructors - They must have different Parameters list

Md. Mamun Hossain
B. Sc. (Engg.) & M. Sc. (Thesis) in CSE, SUST
Asst. Professor, Dept. of CSE , BAUST

Polymorphism : Overloading

- **Constructor Overloading -** With **Constructor overloading**, multiple constructor can have the same name with different (number&/type) parameters.
- **Method Overloading -**With **method overloading**, multiple methods can have the same name with different (number&/type) parameters.

```
class A{
   A(){ … }
   A(String n){ … }

     void show(){ … }
     void show(String n){ … }
     int  show (int a, int b){ … }
}
```

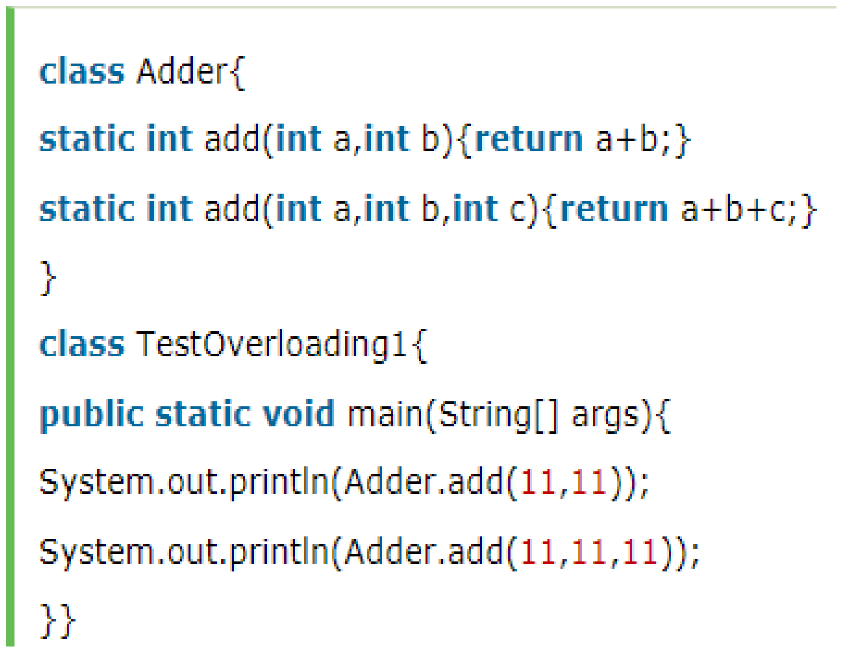
Constructors Overloading

Methods Overloading

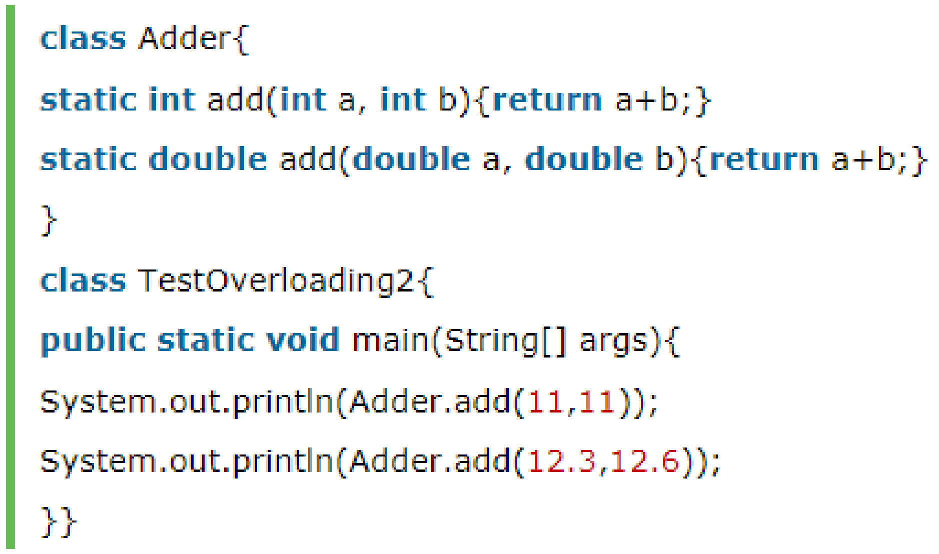
Note: Multiple methods can have the same name as long as the number and/or type of parameters are different.

Md. Mamun Hossain
B. Sc. (Engg.) & M. Sc. (Thesis) in CSE, SUST
Asst. Professor, Dept. of CSE , BAUST

Polymorphism : Overloading - Example



```
class Adder{
static int add(int a,int b){return a+b;}
static int add(int a,int b,int c){return a+b+c;}
}
class TestOverloading1{
public static void main(String[] args){
System.out.println(Adder.add(11,11));
System.out.println(Adder.add(11,11,11));
}}
```



```
class Adder{
static int add(int a, int b){return a+b;}
static double add(double a, double b){return a+b;}
}
class TestOverloading2{
public static void main(String[] args){
System.out.println(Adder.add(11,11));
System.out.println(Adder.add(12.3,12.6));
}}
```

Md. Mamun Hossain
B. Sc. (Engg.) & M. Sc. (Thesis) in CSE, SUST
Asst. Professor, Dept. of CSE , BAUST

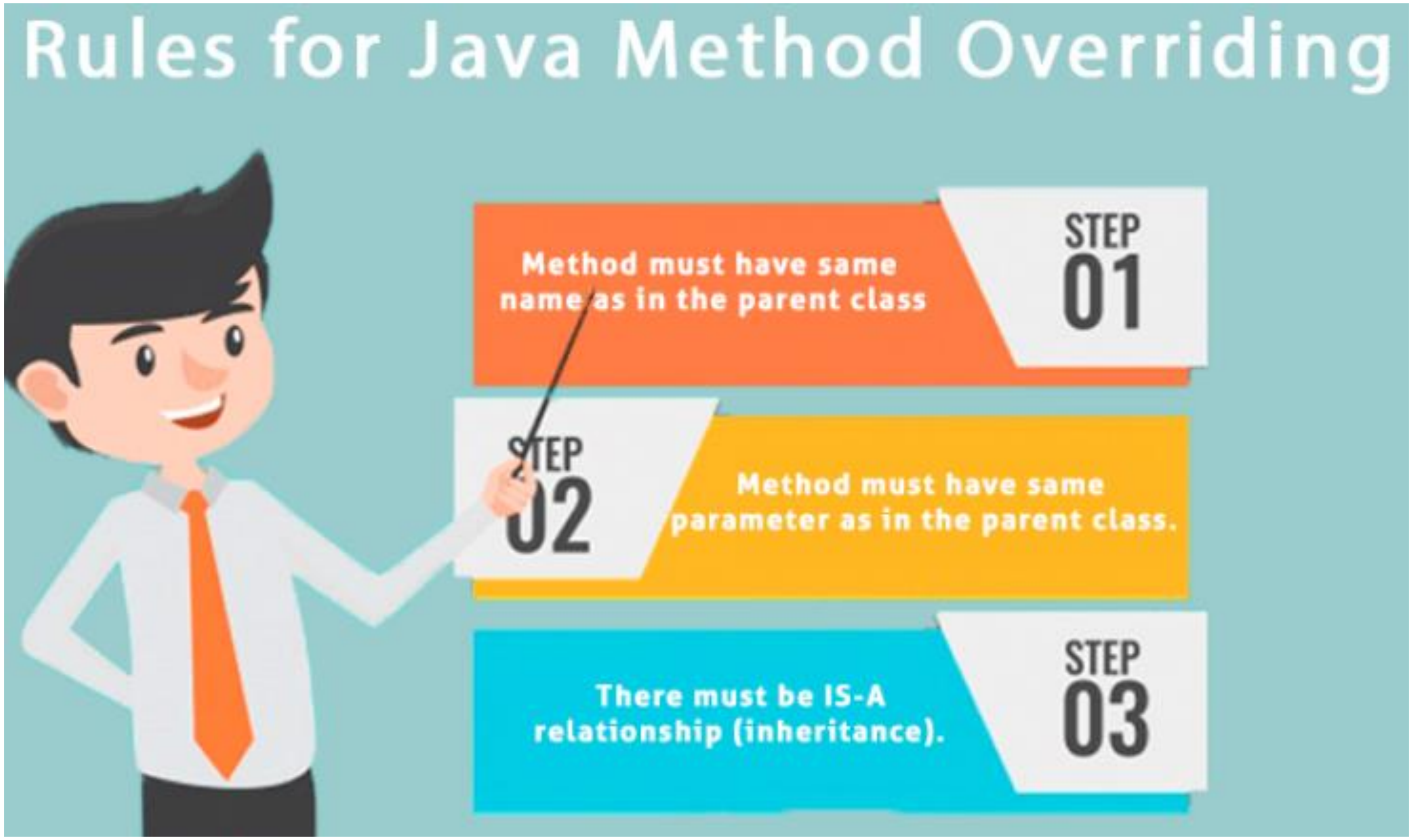
Polymorphism : Overriding

- If subclass (child class) has the same method as declared in the parent class, it is known as **method overriding in Java**.

- 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).

Polymorphism : Overriding - Rules


Rules for Java Method Overriding

STEP 01
Method must have same name as in the parent class

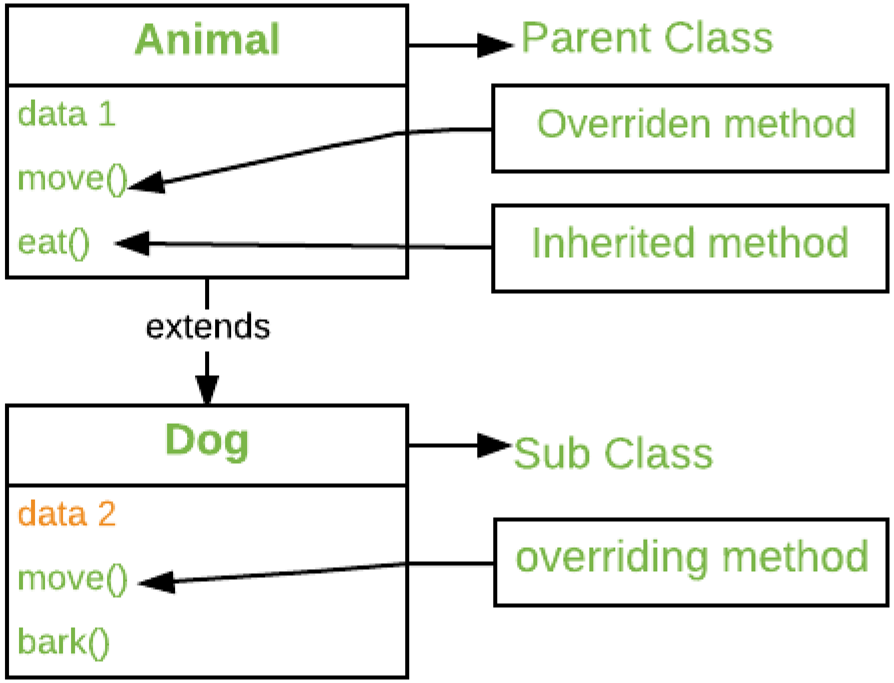
STEP 02
Method must have same parameter as in the parent class.

STEP 03
There must be IS-A relationship (inheritance).

Md. Mamun Hossain
B. Sc. (Engg.) & M. Sc. (Thesis) in CSE, SUST
Asst. Professor, Dept. of CSE , BAUST

Polymorphism : Overriding

- **Overriding**
  - Overriding allows two methods (one is from superclass and another one from subclass) to have the same name and same signatures but different body implementation.
  - Overloading is related to run-time (or dynamic) polymorphism.



Md. Mamun Hossain
B. Sc. (Engg.) & M. Sc. (Thesis) in CSE, SUST
Asst. Professor, Dept. of CSE , BAUST

Polymorphism : Overriding

```
class A{
        A(){ System.out.println(" Default Constructor A "); }
        A(String n){ System.out.println(" Parameterized const. "+n); }
        void show(){ System.out.println(" Hello A ");}
        void show(String n){ System.out.println(" Hello A "+n);}
}
class B extends A{
        B(){ System.out.println(" Default Constructor B "); }
        B(String n){ System.out.println(" Parameterized const. "+n); }
        void show(){ System.out.println(" Hello B "); }
        void show(String n){ System.out.println(" Hello B "+n);}
}
```

Md. Mamun Hossain
B. Sc. (Engg.) & M. Sc. (Thesis) in CSE, SUST
Asst. Professor, Dept. of CSE , BAUST

Polymorphism : Overloading & Overriding

```
class A{
   A() { System.out.println(" Default Constructor A "); }
   A(String n) { System.out.println(" Parameterized const . A "+n); }
            void show() { System.out.println(" Hello A ");}
            void show(String n) { System.out.println(" Hello A "+n);}
            void show(int age) { System.out.println(" Hello  you are "+age);}
}
class B extends A{
        B() { System.out.println(" Default Constructor B "); }
        B(String n) { System.out.println(" Parameterized const. B "+n); }
           void show() { System.out.println(" Hello B "); }
           void show(String n) {System.out.println(" Hello B "+n);}
}
```

Md. Mamun Hossain
B. Sc. (Engg.) & M. Sc. (Thesis) in CSE, SUST
Asst. Professor, Dept. of CSE , BAUST

Polymorphism : Overloading & Overriding

```
class A{
   A() { System.out.println(" Default Constructor A "); }
   A(String n) { System.out.println(" Parameterized const . A "+n); }
            void show() { System.out.println(" Hello A ");}
            void show(String n) { System.out.println(" Hello A "+n);}
            void hello(String n) { System.out.println(" Hello A "+n);}
}
class B extends A{
        B() { System.out.println(" Default Constructor B "); }
        B(String n) { System.out.println(" Parameterized const. B "+n); }
           void show() { System.out.println(" Hello B "); }
           void show(String n) {System.out.println(" Hello B "+n);}
}
```

Md. Mamun Hossain
B. Sc. (Engg.) & M. Sc. (Thesis) in CSE, SUST
Asst. Professor, Dept. of CSE , BAUST

Inheritance & Polymorphism : Overloading & Overriding

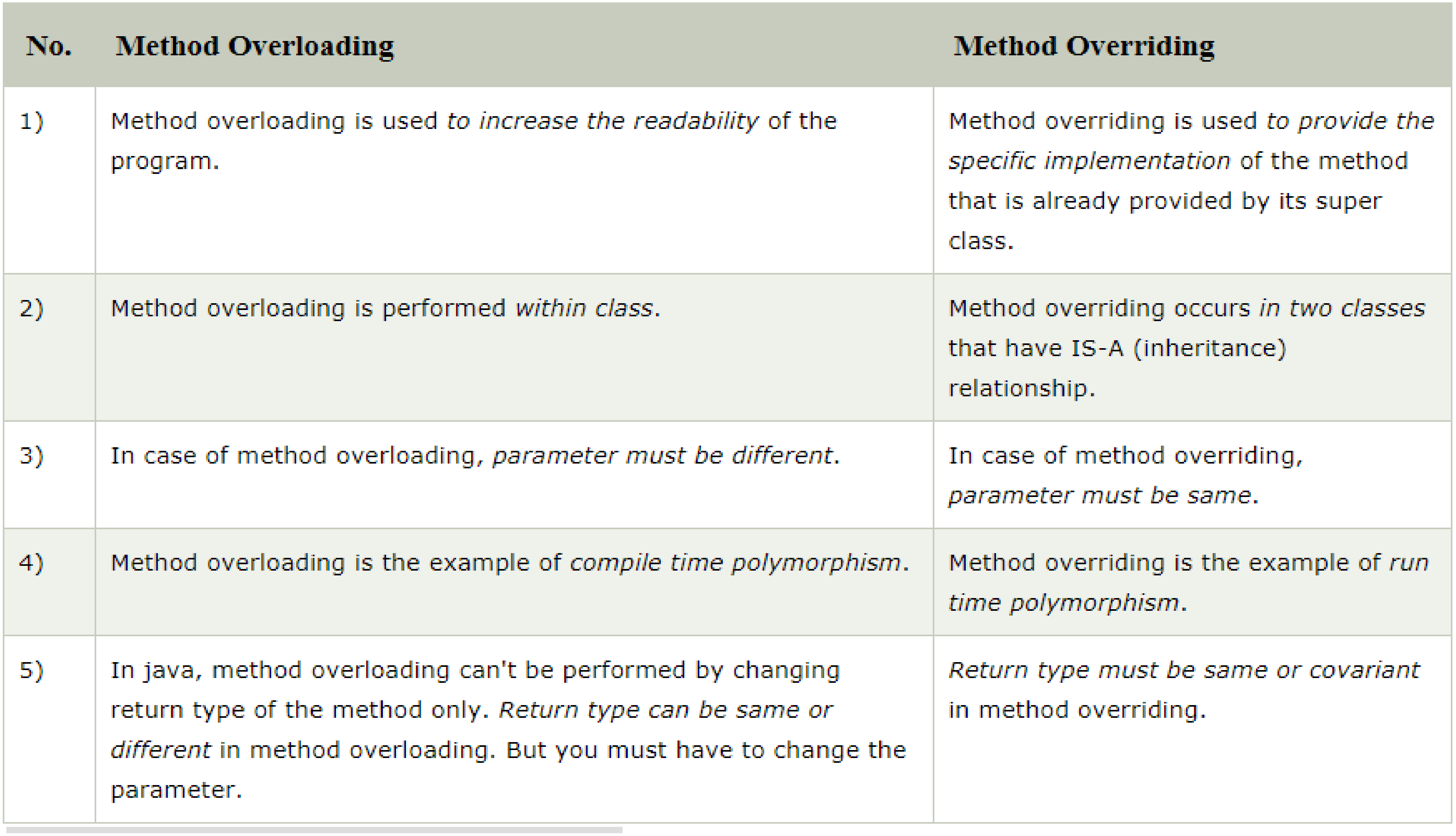
```
class A{
   A(){ System.out.println(" Default Constructor A "); }
   A(String n){ System.out.println(" Parameterized const.  A"+n); }
                  void show(){ System.out.println(" Hello A ");}
                  void show(String n){ System.out.println(" Hello A "+n);}
                  void hello(String n){ System.out.println(" Hello A "+n);}
}
class B extends A{
          B(){ System.out.println(" Default Constructor B "); }
          B(String n){ System.out.println("  Parameterized const. B "+n); }
                   void show(){ System.out.println(" Hello B "); }
                   void show(String n){System.out.println(" Hello B "+n);}
}
```

```
public class Access {
   public static void main(String args[]) {
                A a=new A();
                a.show();
                a.show("Galib");

                B b=new B("Sakib");
                b.show();
                b.show("Daya");
                b.hello("Nisat");
   }
}
```

Output

```
Default Constructor A
Hello A
Hello A Galib
Default Constructor A
Parameterized const. B Sakib
Hello B
Hello B Daya
Hello  A Nisat
```

Overloading Vs. Overriding



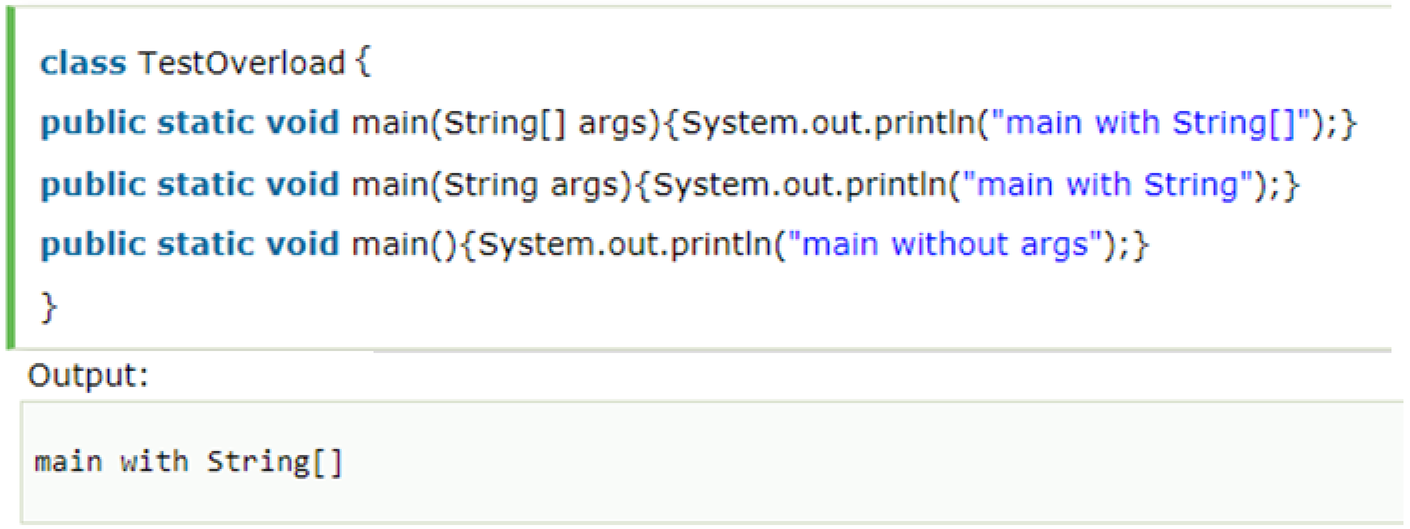
No.	Method Overloading	Method Overriding
1)	Method overloading is used *to increase the readability* of the program.	Method overriding is used *to provide the specific implementation* of the method that is already provided by its super class.
2)	Method overloading is performed *within class*.	Method 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.

Md. Mamun Hossain
B. Sc. (Engg.) & M. Sc. (Thesis) in CSE, SUST
Asst. Professor, Dept. of CSE , BAUST

Can we overload java main() method?

Yes, by method overloading. You can have any number of main methods in a class by method overloading. But JVM calls main() method which receives string array as arguments only.

Let's see the simple example:



```
class TestOverload {
public static void main(String[] args){System.out.println("main with String[]");}
public static void main(String args){System.out.println("main with String");}
public static void main(){System.out.println("main without args");}
}
```

Output:

```
main with String[]
```

Md. Mamun Hossain
B. Sc. (Engg.) & M. Sc. (Thesis) in CSE, SUST
Asst. Professor, Dept. of CSE , BAUST

Inheritance & Polymorphism : Why and When?

Why And When To Use
"Inheritance" and "Polymorphism"?

- **It is useful for code reusability:**
  Reuse attributes and methods of an existing class when you create a new class.

Md. Mamun Hossain
B. Sc. (Engg.) & M. Sc. (Thesis) in CSE, SUST
Asst. Professor, Dept. of CSE , BAUST

Execution Sequence of Constructor

Constructors are executed from ***superclass** to **subclass**.*

Or, we can say,

Constructors are executed in the order they are derived.

```
class A{
		A(){System.out.println("Hello A")}
}
class B extends A{
		B(){System.out.println("Hello B")}
}
class C extends B{
		C(){System.out.println("Hello C")}
}
```

```
class Access{
		public static void main(String args[]){
				C ob=new C()
		}
}
```

Output:

```
Hello A
Hello B
Hello C
```

In a class hierarchy, constructors complete their execution in order of derivation, from superclass to subclass.

Md. Mamun Hossain
B. Sc. (Engg.) & M. Sc. (Thesis) in CSE, SUST
Asst. Professor, Dept. of CSE , BAUST

Execution Sequence of Constructor

```
class A{
        A(){System.out.println(“Hello A”)}
        A(String name){System.out.println(“Hello “+ name)}
}
class B extends A{
        B(){System.out.println(“Hello B”)}
        B(String name){System.out.println(“Hello “+ name)}
}
class C extends B{
        C(){System.out.println(“Hello C”)}
        C(String name){System.out.println(“Hello “+ name)}
}
```

```
class Access{
        public static void main(String args[]){
                C ob1=new C()
                C ob2=new C(“Miraz”)
                B ob3=new B(“Rejoana”)
        }
}
```

In a class hierarchy, constructors complete their execution in order of derivation, from superclass to subclass.

Output:

```
Hello A
Hello B
Hello C
--------------------------
Hello A
Hello B
Hello Miraz
--------------------------
Hello A
Hello Rejoana
```

Execution Sequence of Constructor

The constructors are executed in order of derivation, Why?

If you think about it, it makes sense that constructors complete their execution in order of derivation.

Because a superclass has no knowledge of any subclass, any initialization it needs to perform is separate from and possibly prerequisite to any initialization performed by the subclass.

Therefore, it must complete its execution first.

DMD: Dynamic Method Dispatch

What is Dynamic method dispatch or DMD?

- Dynamic method dispatch is the mechanism by which a call to an overridden method is resolved at run time, rather than compile time.

Dynamic method dispatch is important because this is how Java implements run-time polymorphism.

- We know, "a superclass reference variable can refer to a subclass object"- *Upcasting*

Java uses this fact to resolve calls to overridden methods at run time. How? See next…

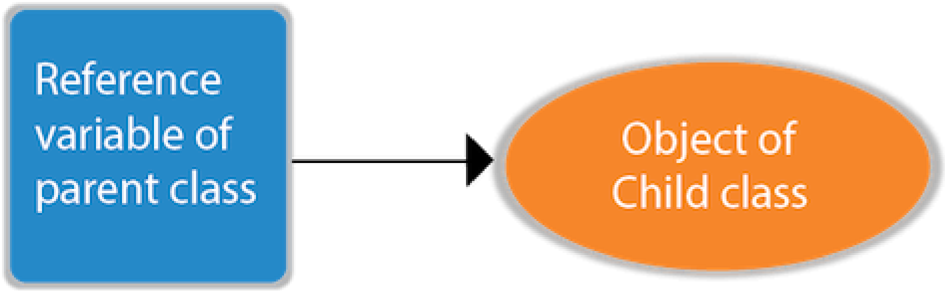
NOTE Readers familiar with C++ or C# will recognize that overridden methods in Java are similar to virtual functions in those languages.

Md. Mamun Hossain
B. Sc. (Engg.) & M. Sc. (Thesis) in CSE, SUST
Asst. Professor, Dept. of CSE , BAUST

DMD: Dynamic Method Dispatch- Upcasting

- **Upcasting:**

  If the reference variable of Parent class refers to the object of Child class, it is known as upcasting. For example



```
class A{}
class B extends A{}
```

```
A a=new B();//upcasting
```

For upcasting, we can use the reference variable of class type or an interface type. For Example:

```
interface I{}
class A{}
class B extends A implements I{}
```

Here, the relationship of B class would be:

```
B IS-A A
B IS-A I
B IS-A Object
```

Since Object is the root class of all classes in Java, so we can write B IS-A Object.

Md. Mamun Hossain
B. Sc. (Engg.) & M. Sc. (Thesis) in CSE, SUST
Asst. Professor, Dept. of CSE , BAUST

DMD: Dynamic Method Dispatch

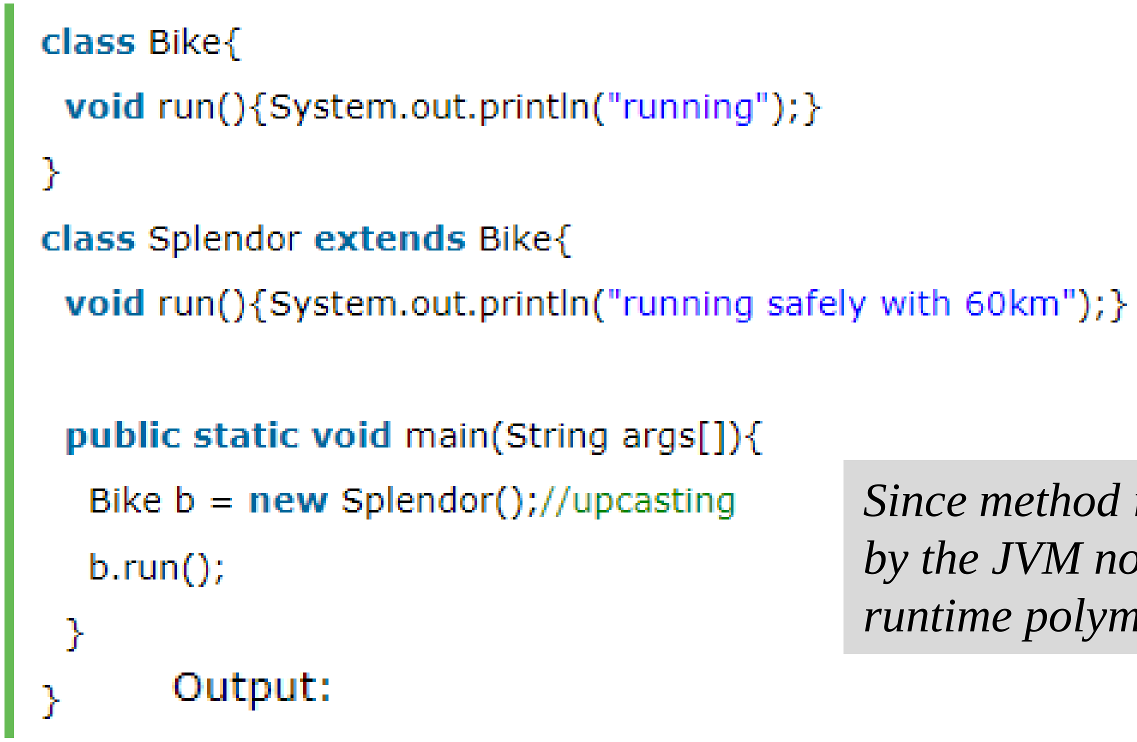
A superclass reference variable can refer to a subclass object.

Java uses this fact to resolve calls to overridden methods at run time. Here is how.

- When an overridden method is called through a superclass reference, Java determines which version of that method to execute based upon the type of the object being referred to at the time the call occurs. Thus, this determination is made at run time.
- When different types of objects are referred to, different versions of an overridden method will be called. In other words, *it is the type of the object being referred to* (not the type of the reference variable) that determines which version of an overridden method will be executed.
- Therefore, if a superclass contains a method that is overridden by a subclass, then when different types of objects are referred to through a superclass reference variable, different versions of the method are executed.

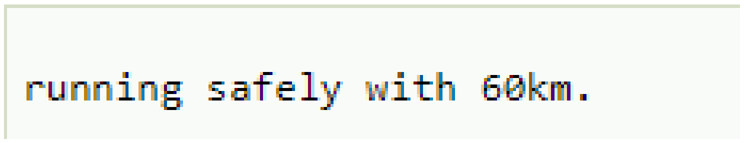
DMD: Dynamic Method Dispatch

- In this process, an overridden method is called through the reference variable of a superclass. The determination of the method to be called is based on the object being referred to by the reference variable.



```
class Bike{
  void run(){System.out.println("running");}
}
class Splendor extends Bike{
  void run(){System.out.println("running safely with 60km");}

  public static void main(String args[]){
    Bike b = new Splendor();//upcasting
    b.run();
  }
}
```

Output:



```
running safely with 60km.
```

Since method invocation is determined by the JVM not compiler, it is known as runtime polymorphism.

Md. Mamun Hossain
B. Sc. (Engg.) & M. Sc. (Thesis) in CSE, SUST
Asst. Professor, Dept. of CSE , BAUST

DMD: Dynamic Method Dispatch

```
class A{
        msg(){System.out.println(“Hello A”)}
}
class B extends A{
        msg(){System.out.println(“Hello B”)}
}
class C extends B{
         msg(){System.out.println(“Hello C”)}
}
```

```
class Access{
public static void main(String args[]){
      A a = new A();
     B b = new B();
      C c = new C();
      A r;
      r = a;  r.msg();
       r = b; r.msg();
       r = c;  r.msg();
     }
}
```

Output:

```
Hello A
Hello B
Hello C
```

Md. Mamun Hossain
B. Sc. (Engg.) & M. Sc. (Thesis) in CSE, SUST
Asst. Professor, Dept. of CSE , BAUST

DMD: Dynamic Method Dispatch

Call to an overridden method is resolved at run-time rather than compile-time is called Dynamic Method Dispatch or DMD.

```
class A{
        msg(){System.out.println(“Hello A”)}
}
class B extends A{
        msg(){System.out.println(“Hello B”)}
}
class C extends B{
         msg(){System.out.println(“Hello C”)}
}
```

Output:

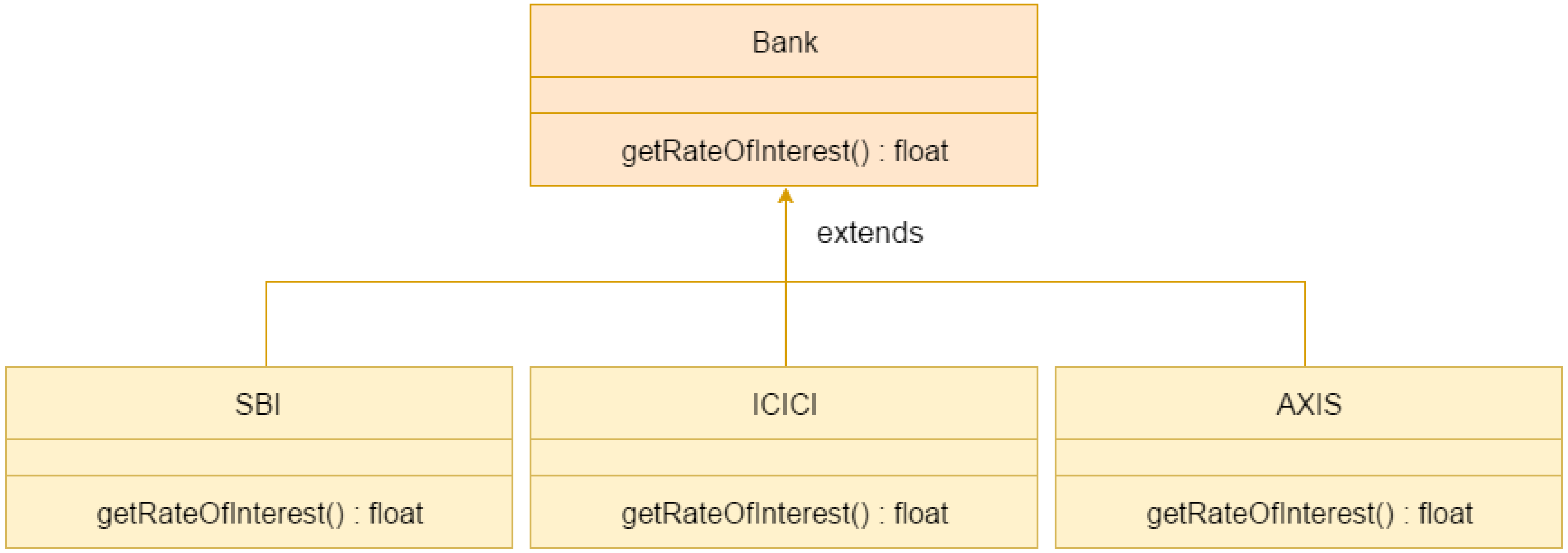
```
Hello A
Hello B
Hello C
```

```
class Access{
public static void main(String args[]){
      A a = new A(); // object of type A
      B b = new B(); // object of type B
      C c = new C(); // object of type C
      A r; // obtain a reference of type A
      r = a; // r refers to an A object
       r.msg(); // calls A's version of msg
       r = b; // r refers to a B object
       r.msg(); // calls B's version of msg
       r = c; // r refers to a C object
        r.msg(); // calls C's version of mag }
}
```

Md. Mamun Hossain
B. Sc. (Engg.) & M. Sc. (Thesis) in CSE, SUST
Asst. Professor, Dept. of CSE , BAUST

DMD: Dynamic Method Dispatch

Consider a scenario where Bank is a class that provides a method to get the rate of interest. However, the rate of interest may differ according to banks. For example, SBI, ICICI, and AXIS banks are providing 8.4%, 7.3%, and 9.7% rate of interest.


Bank
getRateOfInterest() : float
extends
SBI
getRateOfInterest() : float
ICICI
getRateOfInterest() : float
AXIS
getRateOfInterest() : float

Md. Mamun Hossain
B. Sc. (Engg.) & M. Sc. (Thesis) in CSE, SUST
Asst. Professor, Dept. of CSE , BAUST

DMD: Dynamic Method Dispatch

Consider a scenario where Bank is a class that provides a method to get the rate of interest. However, the rate of interest may differ according to banks. For example, SBI, ICICI, and AXIS banks are providing 8.4%, 7.3%, and 9.7% rate of interest.

```
class Bank{
float getRateOfInterest(){return 0;}
}
class SBI extends Bank{
float getRateOfInterest(){return 8.4f;}
}
class ICICI extends Bank{
float getRateOfInterest(){return 7.3f;}
}
class AXIS extends Bank{
float getRateOfInterest(){return 9.7f;}
}
```

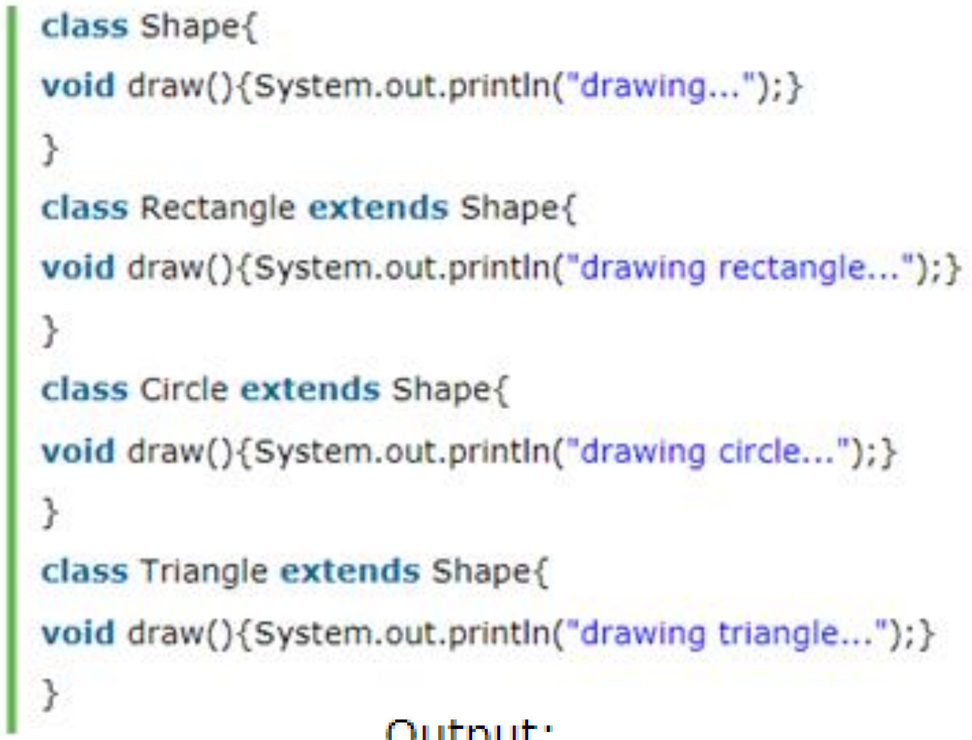
```
class TestPolymorphism{
public static void main(String args[]){
Bank b;
b=new SBI();
System.out.println("SBI Rate of Interest: "+b.getRateOfInterest());
b=new ICICI();
System.out.println("ICICI Rate of Interest: "+b.getRateOfInterest());
b=new AXIS();
System.out.println("AXIS Rate of Interest: "+b.getRateOfInterest());
}
}
```

Output:

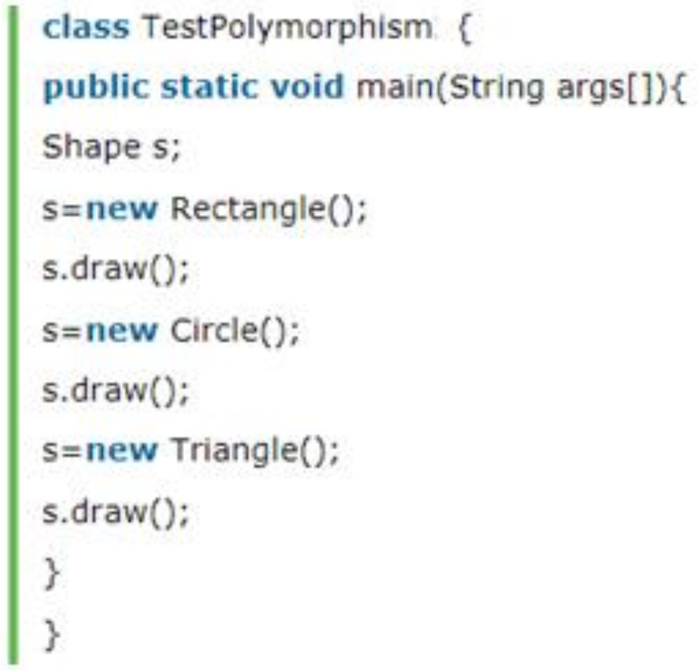
```
SBI Rate of Interest: 8.4
ICICI Rate of Interest: 7.3
AXIS Rate of Interest: 9.7
```

Md. Mamun Hossain
B. Sc. (Engg.) & M. Sc. (Thesis) in CSE, SUST
Asst. Professor, Dept. of CSE , BAUST

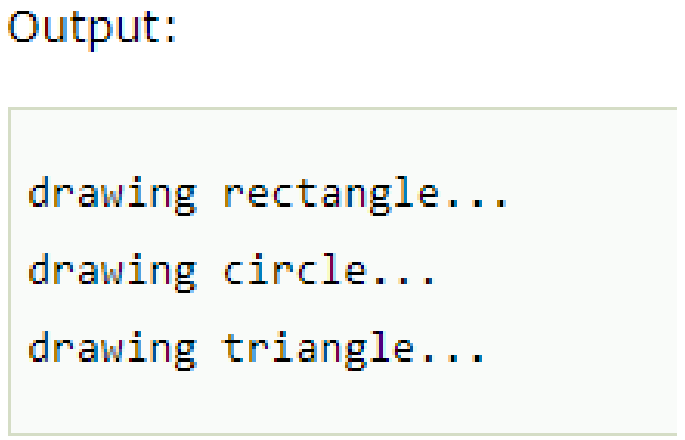
DMD: Dynamic Method Dispatch - Shape



```java
class Shape{
void draw(){System.out.println("drawing...");}
}
class Rectangle extends Shape{
void draw(){System.out.println("drawing rectangle...");}
}
class Circle extends Shape{
void draw(){System.out.println("drawing circle...");}
}
class Triangle extends Shape{
void draw(){System.out.println("drawing triangle...");}
}
```



```java
class TestPolymorphism  {
public static void main(String args[]){
Shape s;
s=new Rectangle();
s.draw();
s=new Circle();
s.draw();
s=new Triangle();
s.draw();
}
}
```

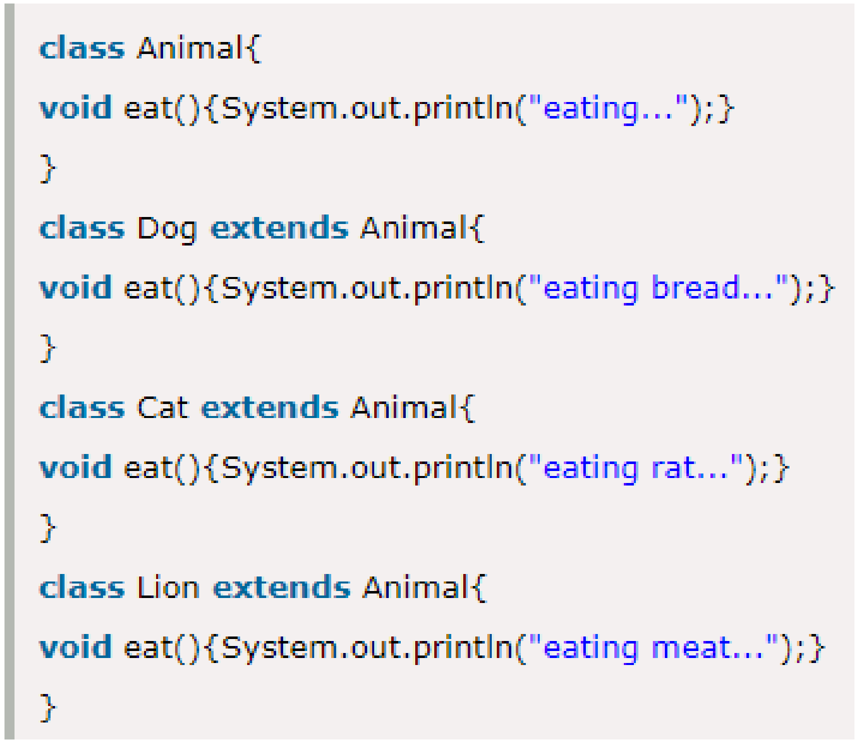


Output:

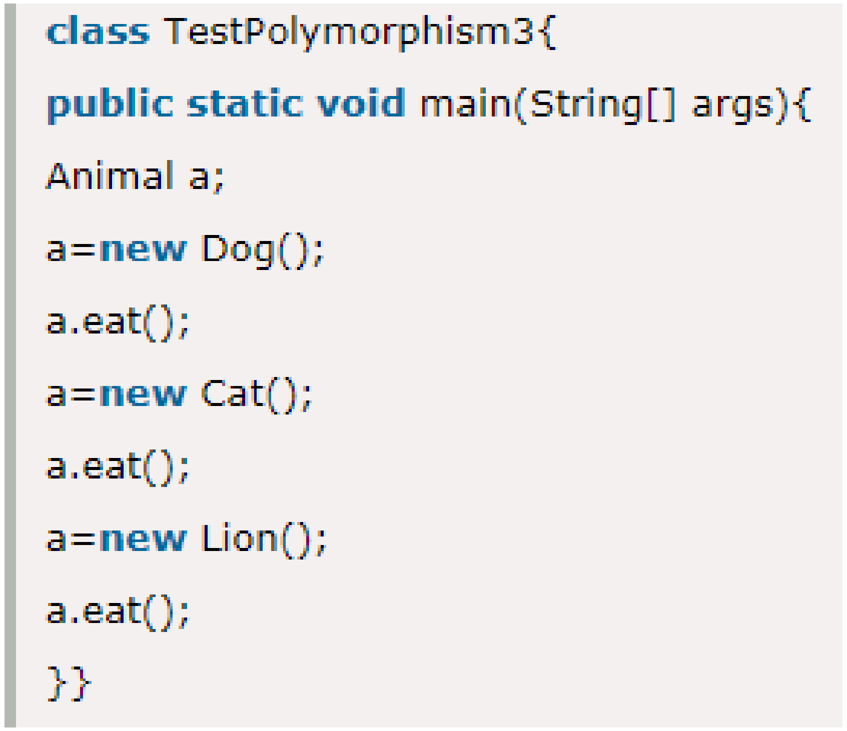
```
drawing rectangle...
drawing circle...
drawing triangle...
```

Md. Mamun Hossain
B. Sc. (Engg.) & M. Sc. (Thesis) in CSE, SUST
Asst. Professor, Dept. of CSE , BAUST

DMD: Dynamic Method Dispatch - Animal

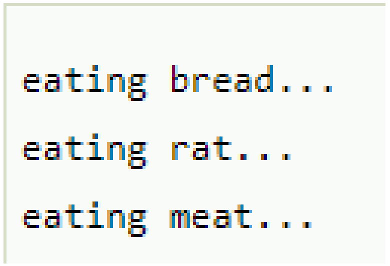


```java
class Animal{
void eat(){System.out.println("eating...");}
}
class Dog extends Animal{
void eat(){System.out.println("eating bread...");}
}
class Cat extends Animal{
void eat(){System.out.println("eating rat...");}
}
class Lion extends Animal{
void eat(){System.out.println("eating meat...");}
}
```



```java
class TestPolymorphism3{
public static void main(String[] args){
Animal a;
a=new Dog();
a.eat();
a=new Cat();
a.eat();
a=new Lion();
a.eat();
}}
```

Output:



```
eating bread...
eating rat...
eating meat...
```

Md. Mamun Hossain
B. Sc. (Engg.) & M. Sc. (Thesis) in CSE, SUST
Asst. Professor, Dept. of CSE , BAUST

The final keyword in Java

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

- ✓ **Variable – final int FILE=1;**
- ✓ **Method – final void min_max();**
- ✓ **Class – final class Box;**

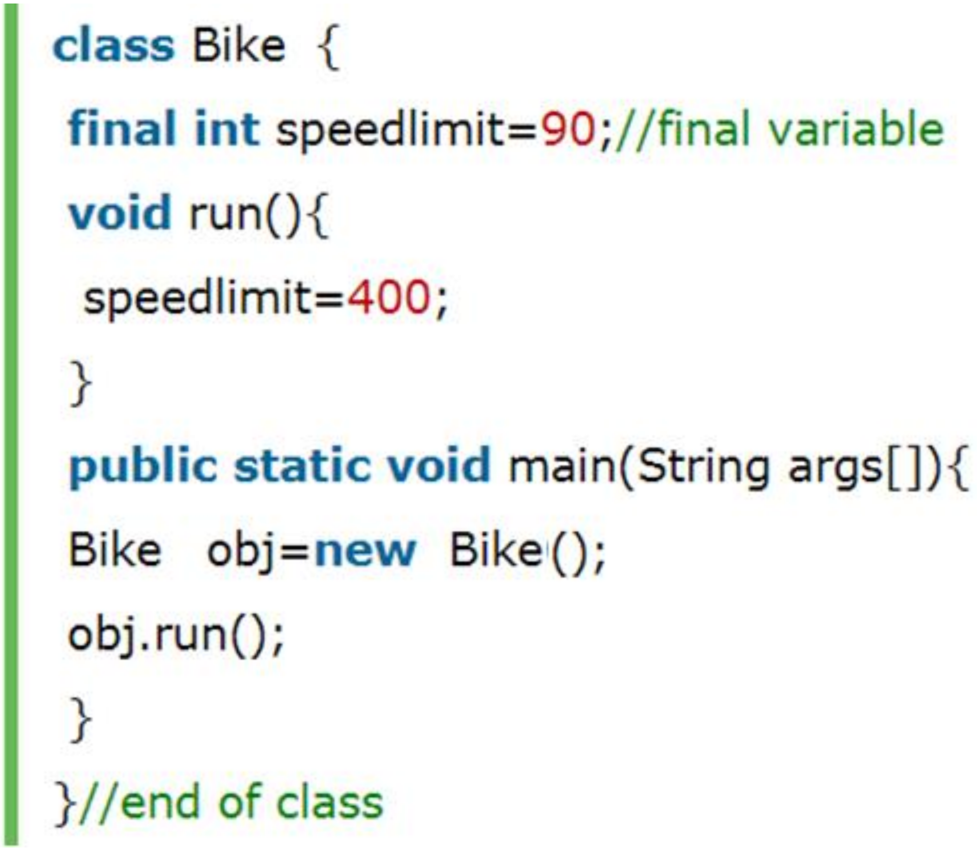
Restrictions:

- ▪ If you make any ***variable*** as final, you ***cannot reassign*** it.
- ▪ If you make any ***method*** as final, you ***cannot override*** it.
- ▪ If you make any ***class*** as final, you ***cannot extend*** it.

Md. Mamun Hossain
B. Sc. (Engg.) & M. Sc. (Thesis) in CSE, SUST
Asst. Professor, Dept. of CSE , BAUST

The final keyword with Variable

- If final variable once assigned a value can never be changed.

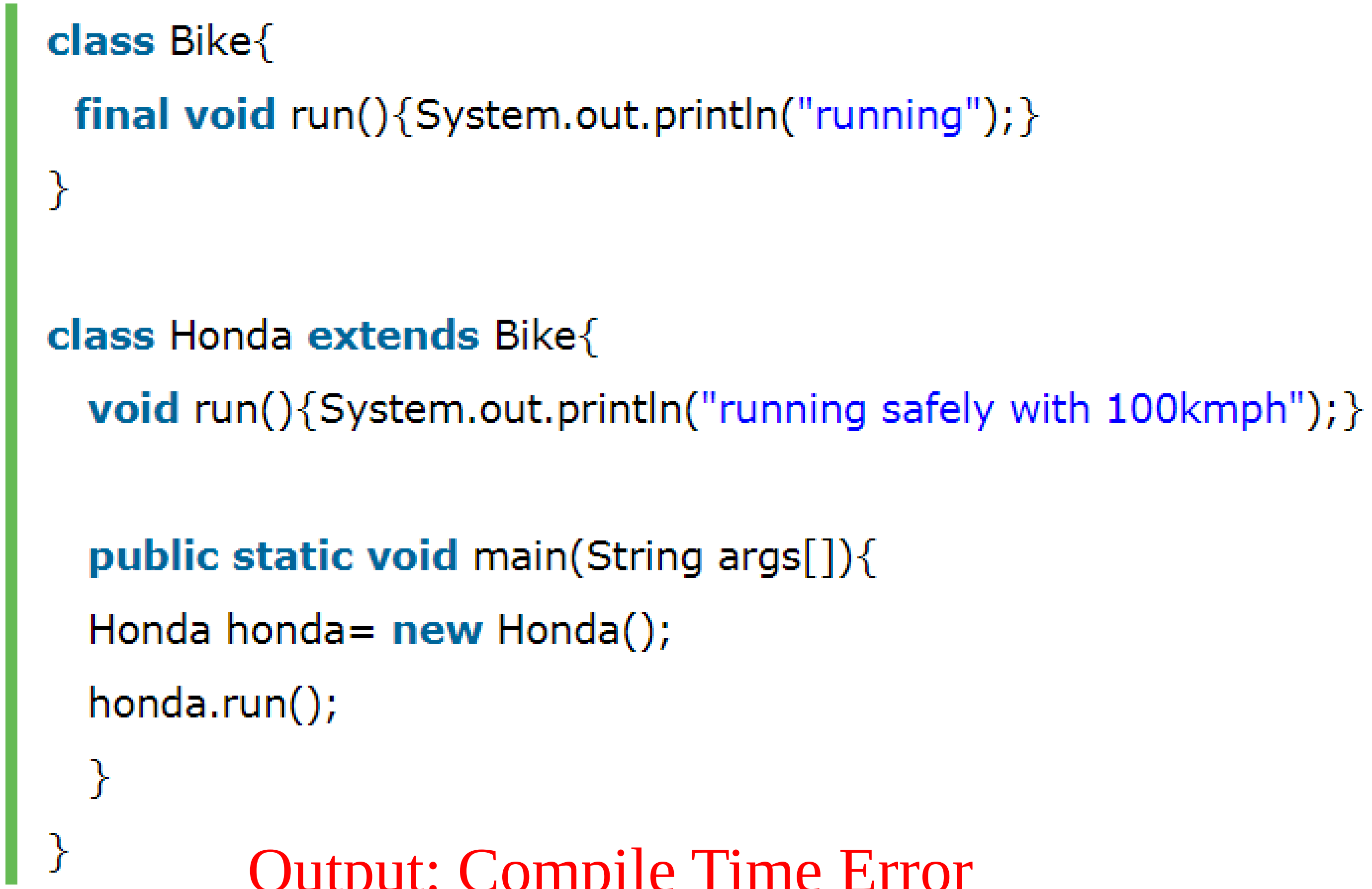


```
class Bike  {
 final int speedlimit=90;//final variable
 void run(){
  speedlimit=400;
 }
 public static void main(String args[]){
 Bike   obj=new  Bike();
 obj.run();
 }
}//end of class
```

Output: Compile Time Error

The final keyword with Method

- If you make any method as final, it cannot be overridden



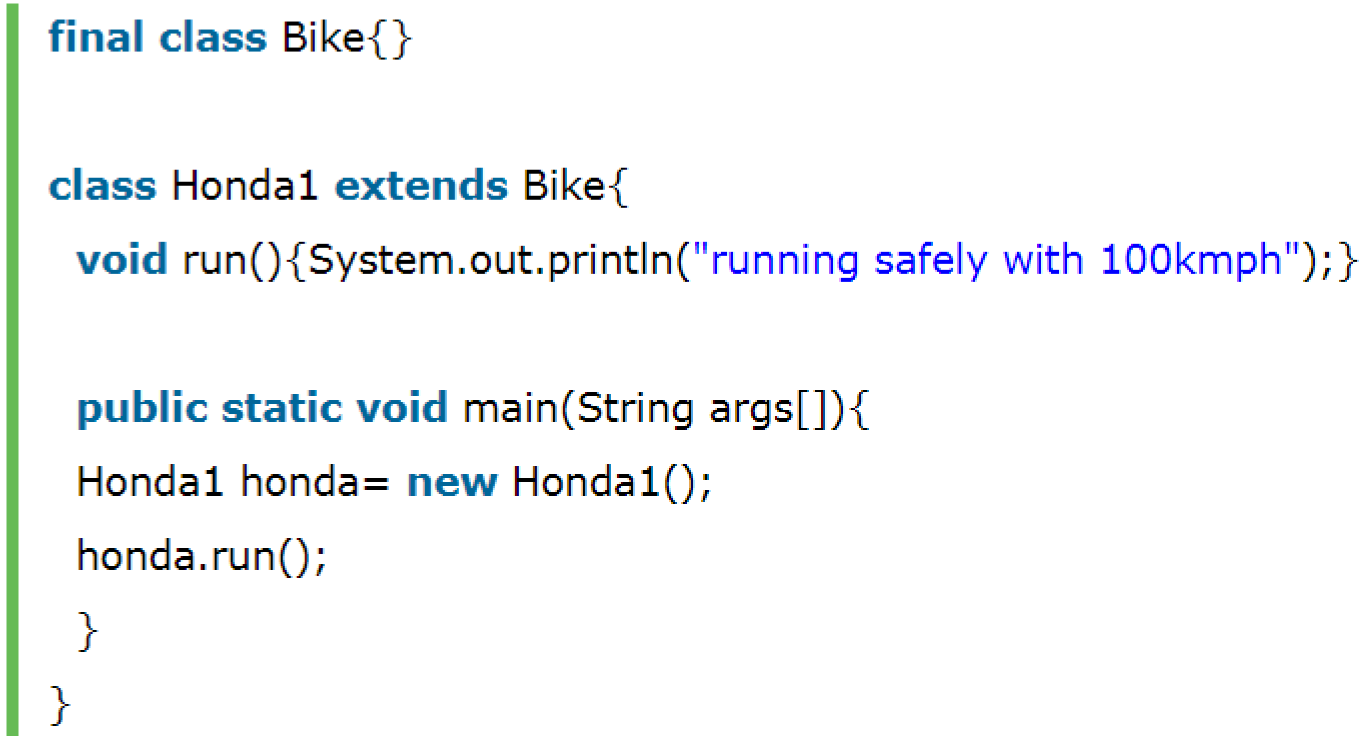
```
class Bike{
  final void run(){System.out.println("running");}
}

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

Md. Mamun Hossain
B. Sc. (Engg.) & M. Sc. (Thesis) in CSE, SUST
Asst. Professor, Dept. of CSE , BAUST

The final keyword with Class

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



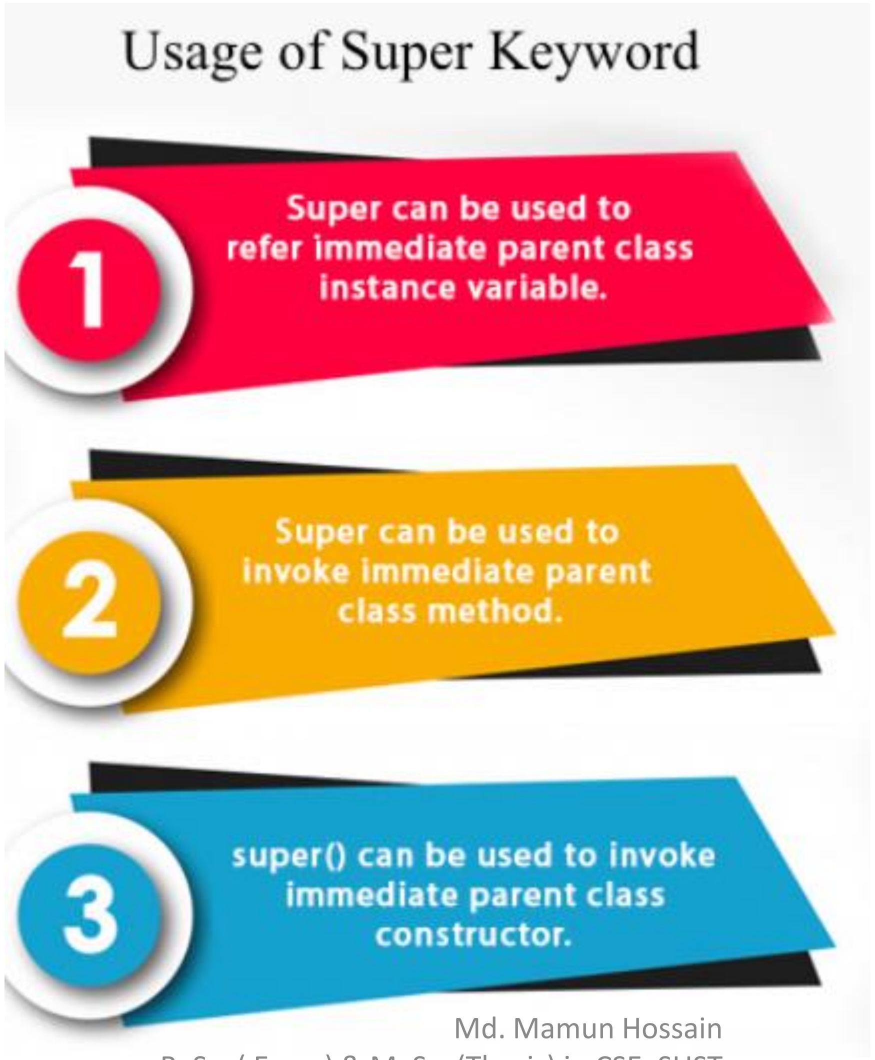
```
final class Bike{}

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

Md. Mamun Hossain
B. Sc. (Engg.) & M. Sc. (Thesis) in CSE, SUST
Asst. Professor, Dept. of CSE , BAUST

Super Keyword in Java

- The **super** keyword in Java is a reference variable which is used to refer ***immediate*** parent class object.
- Whenever you create the instance of subclass, an instance of parent class is created implicitly which is referred by super reference variable.
- **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.


Usage of Super Keyword
1
Super can be used to refer immediate parent class instance variable.
2
Super can be used to invoke immediate parent class method.
3
super() can be used to invoke immediate parent class constructor.

Md. Mamun Hossain
B. Sc. (Engg.) & M. Sc. (Thesis) in CSE, SUST
Asst. Professor, Dept. of CSE , BAUST

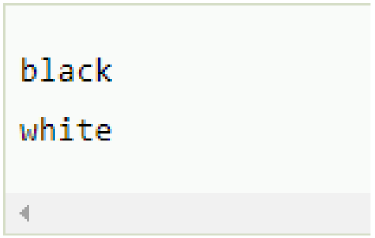
Super Keyword: Member hiding

Super to Access immediate parent class instance variable

- super can be used to refer immediate parent class instance variable. We can use super keyword to access the data member or field of parent class. It is used if parent class and child class have same fields.
- This concept is also referred as member hiding.

```java
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
```

Md. Mamun Hossain
B. Sc. (Engg.) & M. Sc. (Thesis) in CSE, SUST
Asst. Professor, Dept. of CSE , BAUST

Super Keyword: Access Parent Resources

super can be used to invoke parent class method

The super keyword can also be used to invoke parent class methods (Constructors as well). It should be used if subclass contains the same method as parent class. In other words, it is used if method is overridden.

```
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 TestSuper {
public static void main(String args[]){
        Dog d=new Dog();
        d.work();
}}
```

Output:

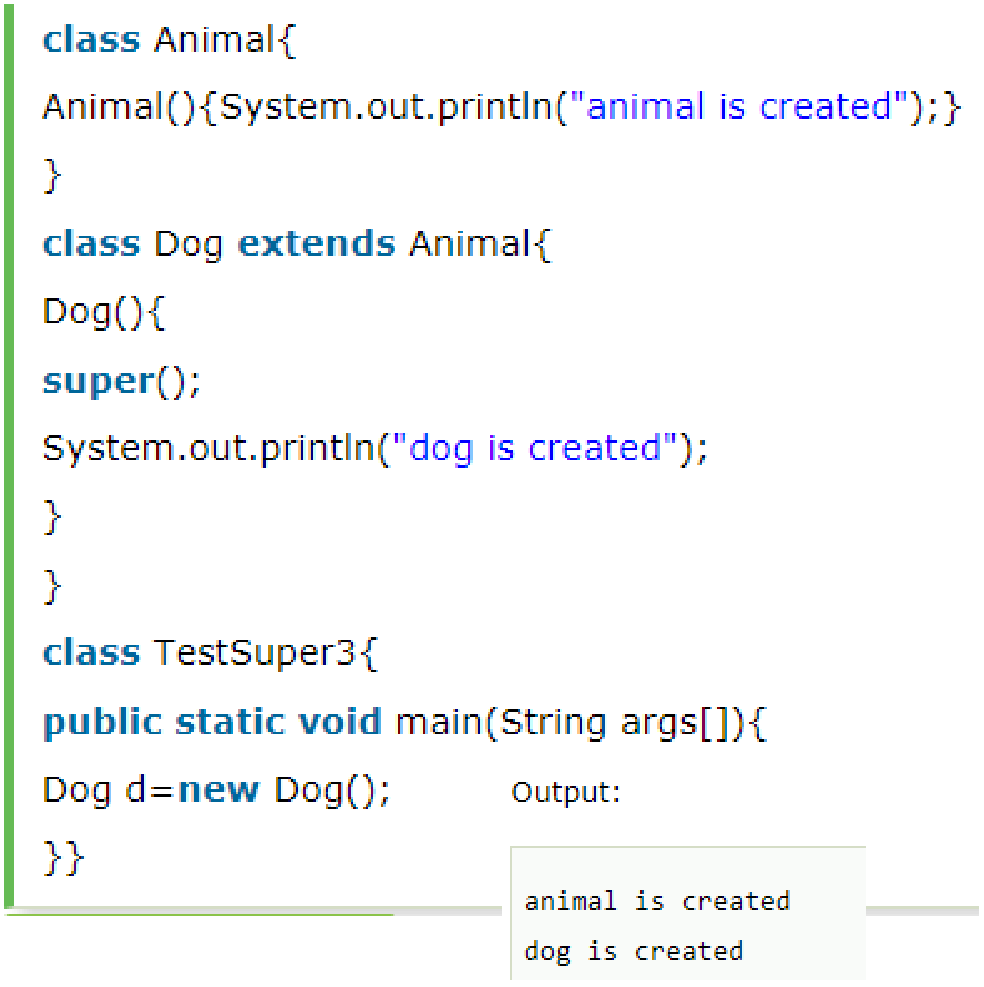
```
eating...
barking...
```

Md. Mamun Hossain
B. Sc. (Engg.) & M. Sc. (Thesis) in CSE, SUST
Asst. Professor, Dept. of CSE , BAUST

Super to Access Parent Class Constructor

Super can be used to invoke parent class constructor

The super keyword can also be used to invoke parent class constructors. You may be wonder as subclass always call a superclass constructor, so why do we need to use super() to call superclass constructor? It may also seems redundant and unnecessary.
But this is specially useful when you use super to call superclass parameterized constructor.



```
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();
}}
```

Output:

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

Md. Mamun Hossain
B. Sc. (Engg.) & M. Sc. (Thesis) in CSE, SUST
Asst. Professor, Dept. of CSE , BAUST

Use of Super to call superclass parameterized constructor

```
class Animal{
        Animal(){System.out.println(“Animal is created");}
        Animal(String name){System.out.println(name + " is created");}
}
class Dog extends Animal{
        Dog(){
                super(“Pupy”)
                System.out.println(“Dog is created");
        }
}
class TestSuper{
        public static void main(String args[]){
                Dog d=new Dog();
        }
}
```

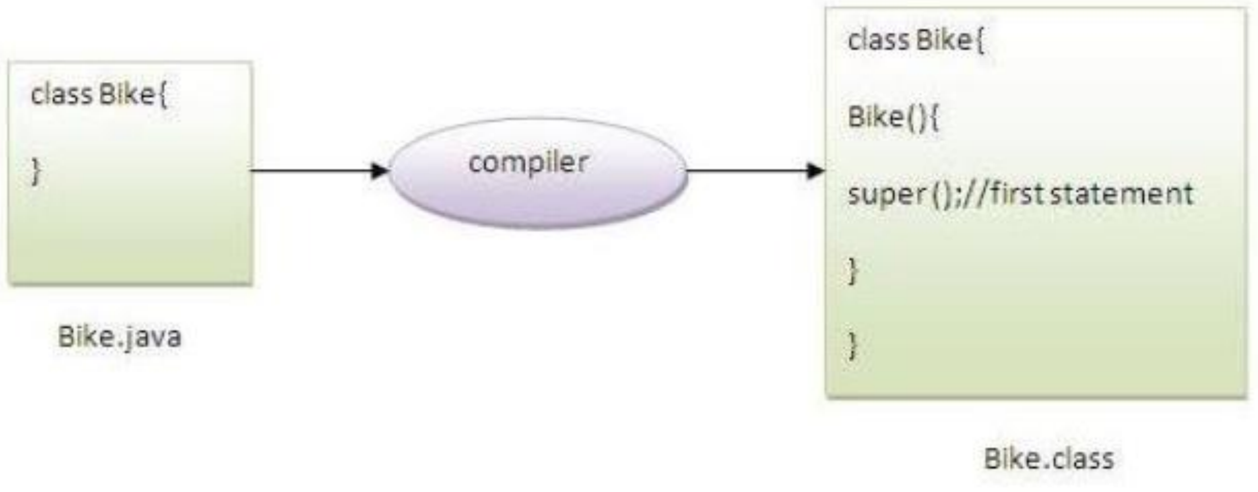
Output:

```
Pupy is created
Dog is created
```

Md. Mamun Hossain
B. Sc. (Engg.) & M. Sc. (Thesis) in CSE, SUST
Asst. Professor, Dept. of CSE , BAUST

Super added automatically by Compiler

Note: super() is added in each class constructor automatically by compiler if there is no super() or this().



```
class Bike{

}
```

```
class Bike{

Bike(){

super();//first statement

}

}
```

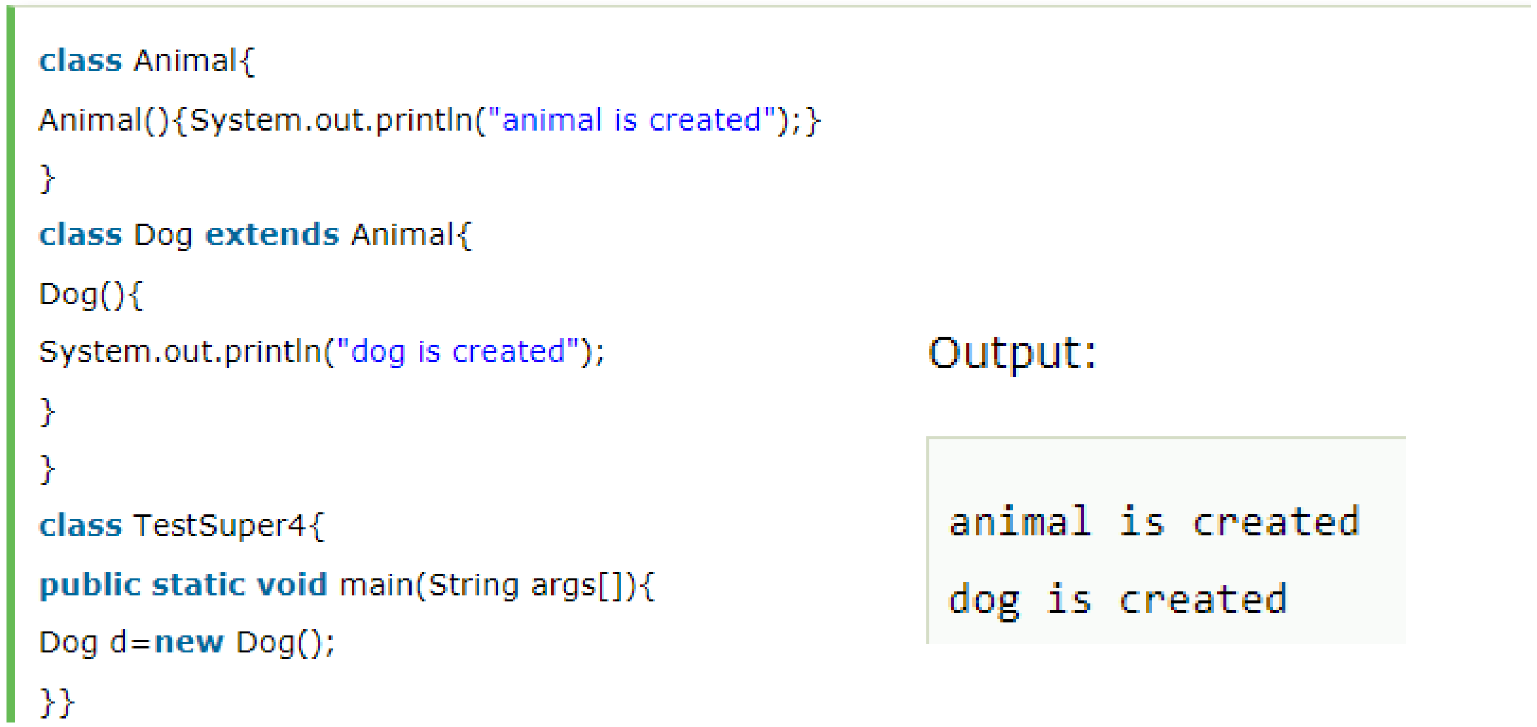
NB: ***super() should be the first statement in the constructor.***

Md. Mamun Hossain
B. Sc. (Engg.) & M. Sc. (Thesis) in CSE, SUST
Asst. Professor, Dept. of CSE , BAUST

super() as the first statement

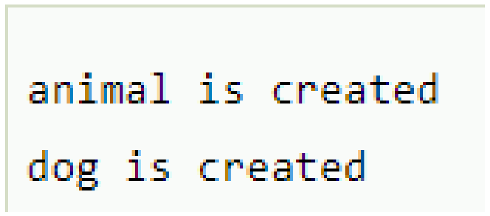
As we know well that default constructor is provided by compiler automatically if there is no constructor. But, it also adds super() as the first statement.

Another example of super keyword where super() is provided by the compiler implicitly.



```java
class Animal{
Animal(){System.out.println("animal is created");}
}
class Dog extends Animal{
Dog(){
System.out.println("dog is created");
}
}
class TestSuper4{
public static void main(String args[]){
Dog d=new Dog();
}}
```

Output:



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

Md. Mamun Hossain
B. Sc. (Engg.) & M. Sc. (Thesis) in CSE, SUST
Asst. Professor, Dept. of CSE , BAUST

Appendix

- **Inheritance: Has-A Relationship**
  - A practical real world example
- **Super: call to superclass parameterized Constructor**
  - A practical real world example
- **Final: Assignment**
  - In assignment you have to solve two problems

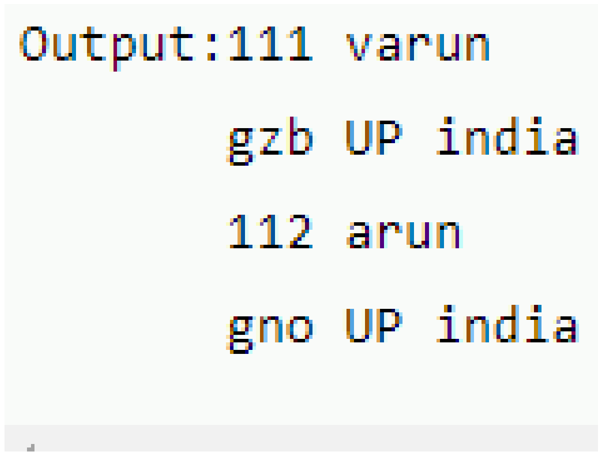
Inheritance: Has-A Relationship

```
public class Address {
        String city, state, country;
        public Address(String ct, String st, String cy) {
                city = ct;
                state = st;
                country = cy;
        }
}
```

```
class Employee{
      int id;
      String name;
      Address add;
    public Emp(int id, String n, Address a) {
         id = id;
         name = n;
         add=a;
    }
   void display(){
         System.out.println(id+" "+name);
         System.out.println(add.city+" "+
        add.state+ " "+add.country);
    }
```

- **A real world Example**

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

 Address ad1=new Address("gzb","UP","india");
 Address ad2=new Address("gno","UP","india");

 Emp e1=new Emp(111,"varun",ad1);
 Emp e2=new Emp(112,"arun",ad2);

                e.display();
                e2.display();
  }
}
```



```
Output:111 varun
       gzb UP india
       112 arun
       gno UP india
```

Super: A real world Example

```
class Person{
        int id;
        String name;
        Person(int id,String name){
                this.id=id;
                this.name=name;
        }
}
class Emp extends Person{
        float salary;
        Emp(int id, String name, float salary){
                super(id,name);//reusing parent constructor
                this.salary=salary;
        }
        void display(){System.out.println(id+" "+name+" "+salary);}
}
```

```
class TestSuper{
        public static void main(String[] args){
                Emp e1=new Emp(1,"ankit",45000f);
                e1.display();
        }
}
```

Output:

```
1 ankit 45000
```

Md. Mamun Hossain
B. Sc. (Engg.) & M. Sc. (Thesis) in CSE, SUST
Asst. Professor, Dept. of CSE , BAUST

The final keyword : Assignment

- What will happen if we declare a variable, a method and a class with final keyword? Explain with an suitable code example.
- The following java program contains some bugs regarding final keyword. Explain the cause of bug and debug it (rewrite the program so that it become bug free).

```
final class A{
        final int speed=90;
        final void show( int s){  speed=s ; }
 }
class B extends A{
        void show(){  System.out.println("Nothing");}
}
class Access{
        public static void main(String args[]){
                A a = new B();  a.show();
        }
 }
```

Md. Mamun Hossain
B. Sc. (Engg.) & M. Sc. (Thesis) in CSE, SUST
Asst. Professor, Dept. of CSE , BAUST