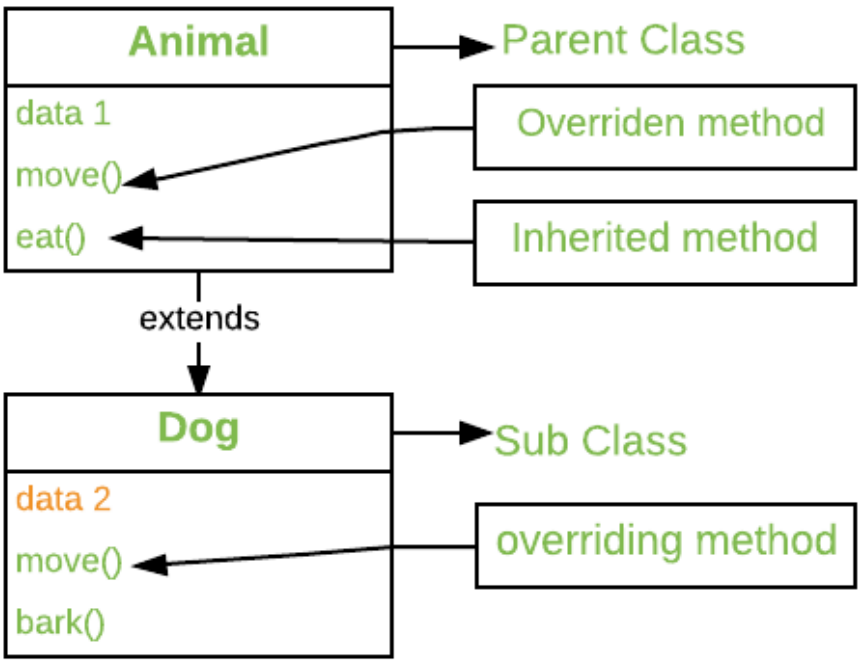
Birla Institute of Technology & Science, Pilani – K. K. Birla Goa campus
Object Oriented Programming
First Semester 2017-2018
Lab-05 (Inheritance)

Objective: This reading material contains some of the concepts that are useful for the lab on inheritance.

1. Method Overriding:

In any object-oriented programming language, *Overriding* is a feature that allows a subclass or child class to provide a specific implementation of a method that is already provided by one of its superclasses or parent classes. When a method in a subclass has the same name, same parameters or signature and same return type (or subtype) as a method in its superclass, then the method in the subclass is said to ***override*** the method in the super-class.



The benefit of overriding: It allows us to define a behavior that's specific to the subclass type, which means a subclass can implement a parent class method based on its requirement.

In object-oriented terms, overriding means to override the functionality of an existing method.

Example:

```
class Animal {
   public void move() {
      System.out.println("Animals can move");
   }
}
```

```
class Dog extends Animal {
   public void move() {
      System.out.println("Dogs can walk and run");
   }
   public void bark() {
      System.out.println("Dogs can bark");
   }

}

public class TestDog {
   public static void main(String args[]) {
      Animal a = new Animal(); // Animal reference and Animal object
      Animal b = new Dog();   // Animal reference but Dog object

      a.move();   // runs the method in Animal class
      b.move();   // runs the method in Dog class
      b.bark();    //bark method call
   }
}
```

Output:

This program will produce the following result without bark method call:

Animals can move
Dogs can walk and run

With bark method call, It generates Compile Time Error:

```
TestDog.java:26: error: cannot find symbol
     b.bark();
      ^
  symbol:   method bark()
location: variable b of type Animal
1 error
```

Reason:

In the above example, it can see that even though *b* is a type of Animal, it runs the *move* method in the Dog class. The reason for this is; At compile time, the check is made on the reference type. However, in the runtime, JVM figures out the object type and would run the method that belongs to that particular object.

Therefore, in the above example, the program will compile properly since Animal class has the method move. Then, at the runtime, it runs the method specific for that object. However, *b*'s reference type Animal doesn't have a method by the name of bark. So, it generates a compile time error.

Rules for Method Overriding:

1. The argument list should be the same as that of the overridden method.

2. The return type should be the same or a subtype of the return type declared in the original overridden method in the superclass.

3. The access level cannot be more restrictive than the overridden method's access level. For example: If the superclass method is declared public then the overriding method in the subclass cannot be either private or protected.

More restrictive to less restrictive Access Modifiers:

private -> default -> protected -> public

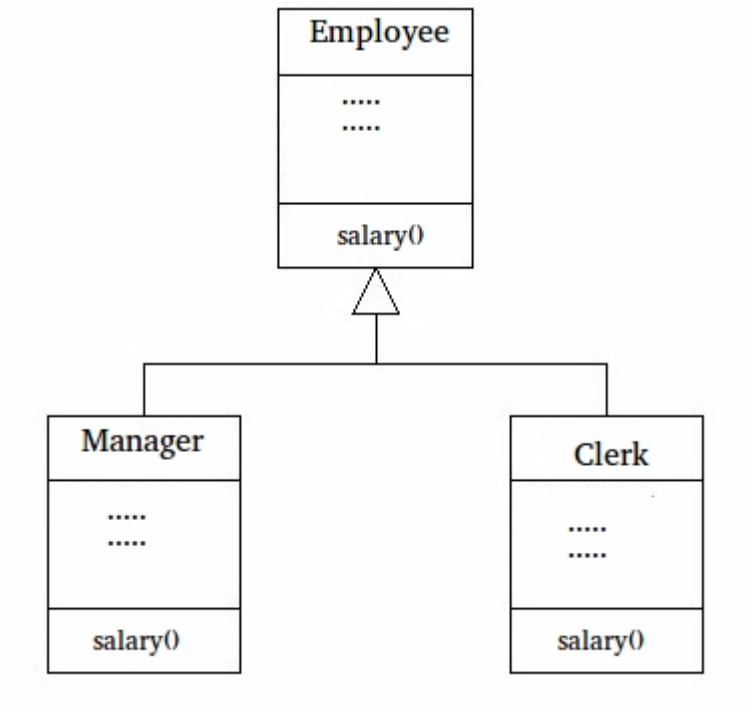
4. private, final, static methods cannot be overridden.

2. Overloading vs. Overriding:

1. Overloading is about the same method have different signatures. Overriding is about the same method, same signature but different classes connected through inheritance.

2. Overloading is an example of compiler-time polymorphism and overriding is an example of runtime polymorphism.

3. Overriding and Inheritance



A Simple Java program to demonstrate the application of Overriding in Java:

```
// Base Class
class Employee
{
     public static int base = 10000;
     int salary()
     {
          return base;
     }
}

// Inherited class
class Manager extends Employee
{
     // This method overrides show() of Parent
     int salary()
     {
          return base + 20000;
     }
}
```

```
// Inherited class
class Clerk extends Employee
{
     // This method overrides show() of Parent
     int salary()
     {
          return base + 10000;
     }
}

// Driver class
class Main
{
     public static void main(String[] args)
     {
          Employee obj1 = new Manager();
     /*here depending on the runtime object corresponding method is called*/
          System.out.print("Manager's salary : ");
          System.out.println(obj1.salary());

          Employee obj2 = new Clerk();
          System.out.print("Clerk's salary : ");
          System.out.println(obj2.salary());
     }
}
```

Output:
Manager's salary : 30000
Clerk's salary : 20000

4. Super Keyword

The super keyword in Java is a reference variable that is used to refer parent class objects. The keyword “super” came into the picture with the concept of Inheritance. Super keyword is used mainly in the following contexts:

1. Use of super with variables:

This scenario occurs when a derived class and base class has the same data members.

2. Use of super with methods:

This is used when we want to the call the parent class method. Whenever a parent and child class have same named methods, super keyword is used to resolve ambiguity.

3. Use of super with Constructors:

super keyword can also be used to access the parent class constructor. One more important thing is that ‘super’ can call both parametric as well as non-parametric constructors depending upon the situation.

Example Program to demonstrate the usage of super keyword:

```
/* superclass Person */

class Person

{

     Person()

     {
          System.out.println("Person class Constructor");
     }
    void message()
     {
          System.out.println("This is person class");
     }
}

/* subclass Student extending the Person class */
class Student extends Person
{
     Student()
     {
        // invoke or call parent class constructor
        super();
        System.out.println("Student class Constructor");
     }
        void message()
     {
```

```
        System.out.println("This is student class");
    }
    // Note that display() is only in Student class
    void display()
    {
        // will invoke or call current class message() method
        message();
        // will invoke or call parent class message() method
        super.message();
    }
}
/* Driver program to test*/
class Test
{
    public static void main(String[] args)
    {
        Student s = new Student();
        s.display();
    }
}
```

Output:

Person class Constructor

Student class Constructor

This is student class

This is person class

5. Java String equals():

The Java String equals() method compares the two given strings based on the content of the string. If characters are not matched, it returns *false*. If all characters are matched, it returns *true*. The String equals() method overrides the equals() method of Object class.

Example:

```
public class EqualsExample{

     public static void main(String args[]){
          String s1="javaworld";
          String s2="javaworld";
          String s3="JAVATWORLD";
          String s4="python";
          System.out.println(s1.equals(s2));
          //true because content and case is same
          System.out.println(s1.equals(s3));
          //false because case is not same
          System.out.println(s1.equals(s4));
          //false because content is not same
          }

     }
```

Output:

true

false

false