Department of Software Engineering
Mehran University of Engineering and Technology, Jamshoro

Course: SWE121 – Object Oriented Programming			
Instructor	Mariam Memon	**Practical/Lab No.**	08
Date		**CLOs**	3
Signature		**Assessment Score**	

Topic	**Implementing the concepts of polymorphism**
Objectives	To demonstrate the concept of polymorphism. To achieve method overloading and overriding and to understand interface and abstract classes

Lab Discussion: Theoretical concepts and Procedural steps

Polymorphism

Polymorphism is the ability of an object to take on many forms. The most common use of polymorphism in OOP occurs when a parent class reference is used to refer to a child class object.

Any Java object that can pass more than one IS-A test is considered to be polymorphic. 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.

It is important to know that the only possible way to access an object is through a reference variable. A reference variable can be of only one type. Once declared, the type of a reference variable cannot be changed.

The reference variable can be reassigned to other objects provided that it is not declared final. The type of the reference variable would determine the methods that it can invoke on the object.

A reference variable can refer to any object of its declared type or any subtype of its declared type. A reference variable can be declared as a class or interface type.

We can perform polymorphism in java by method overloading and method overriding.

Method Overloading

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

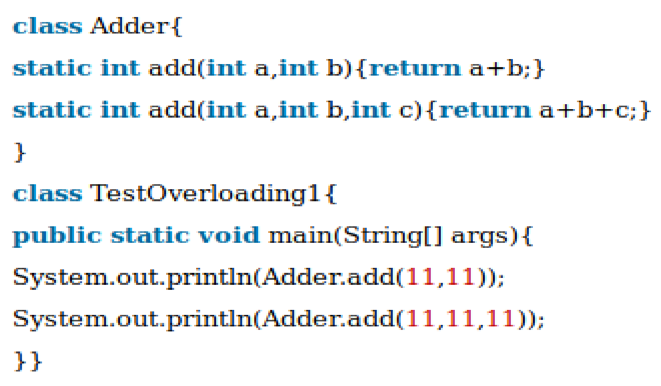
If we have to perform only one operation, having same name of the methods increases the readability of the program.

Suppose you have to perform addition of the given numbers but there can be any number of arguments, if you write the method such as a(int,int) for two parameters, and b(int,int,int) for three parameters then it may be difficult for you as well as other programmers to understand the behavior of the method because its name differs.

There are three ways to overload the method in java

1. By changing number of arguments
2. By changing the data type
3. By changing the sequence of parameters

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));
}}
```

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

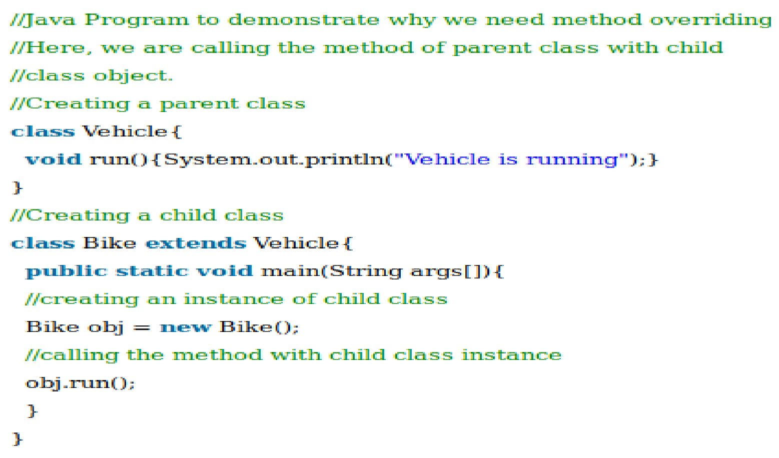
- Method overriding is used to provide the specific implementation of a method which is already provided by its superclass.
- Method overriding is used for runtime polymorphism

Rules for Java Method Overriding

1. The method must have the same name as in the parent class
2. The method must have the same parameter as in the parent class.

3. There must be an IS-A relationship (inheritance).

Example:



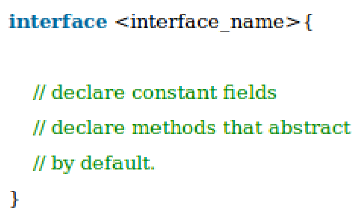
```
//Java Program to demonstrate why we need method overriding
//Here, we are calling the method of parent class with child
//class object.
//Creating a parent class
class Vehicle{
  void run(){System.out.println("Vehicle is running");}
}
//Creating a child class
class Bike extends Vehicle{
  public static void main(String args[]){
  //creating an instance of child class
  Bike obj = new Bike();
  //calling the method with child class instance
  obj.run();
  }
}
```

Interfaces

An interface in java is a blueprint of a class. It has static constants and abstract methods. The interface in Java is *a mechanism to achieve abstraction*. There can be only abstract methods in the Java interface, not method body. It is used to achieve abstraction and multiple inheritance in Java. In other words, you can say that interfaces can have abstract methods and variables. It cannot have a method body. Java Interface also represents the IS-A relationship and cannot be instantiated just like the abstract class.

Since **Java 8**, we can have default and static methods in an interface. Since **Java 9**, we can have private methods in an interface.

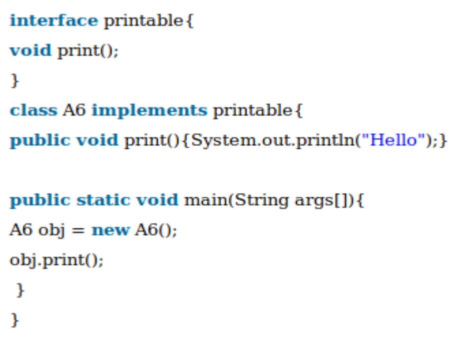
Syntax:



```
interface <interface_name>{

    // declare constant fields
    // declare methods that abstract
    // by default.
}
```

Example:



```
interface printable{
void print();
}
class A6 implements printable{
public void print(){System.out.println("Hello");}

public static void main(String args[]){
A6 obj = new A6();
obj.print();
 }
}
```

Lab Tasks

Note: Make use of packages in creating the following programs

Task 1: Create an interface *AdvancedArithmetic* which contains a method signature *int sumOfFactors(int n)*. You need to write a class called MyCalculator which implements the interface.
sumOfFactors function just takes an integer as input and return the sum of all its factors. For example factors of 6 are 1, 2, 3 and 6, so *sumOfFactors* should return 12. The value of n will be at most 1000.

Task 2: Consider the following class:

```
class Sports{
    String getName(){
        return "Generic Sports";
    }
    void getNumberOfTeamMembers(){
        System.out.println( "Each team has n players in " + getName() );
    }
}
```

Make two classes Cricket and Soccer that inherit the Sports class and override both the methods present in this class.

Task 3: Create a program that demonstrates how final keyword stops method overriding