

**Department of Computer Science**

**SUMMER 2018**

# **CS1421 – Object Oriented Programming Lab**

# **Lab 6**

**Instructor:**

Aman Farooq

**Lab 6**

**Method Overloading and Overriding**

**Objective**

After completing this lab, the students should be able to

* Understand constructor chaining
* Understand the usage and implementation of method overloading and over riding
* Understand usage and implementation of static variables and static methods

**Constructor Chaining:**

A constructor may invoke an overloaded constructor or its superclass constructor. If neither is invoked explicitly, the compiler automatically puts super() as the first statement in the constructor. For example:

public SubClassName() {

// some statements

}

**Equivalent to**

public SubClassName() {

super();

// some statements

}

**OR**

public SubClassName(double d) {

// some statements

}

**Equivalent to**

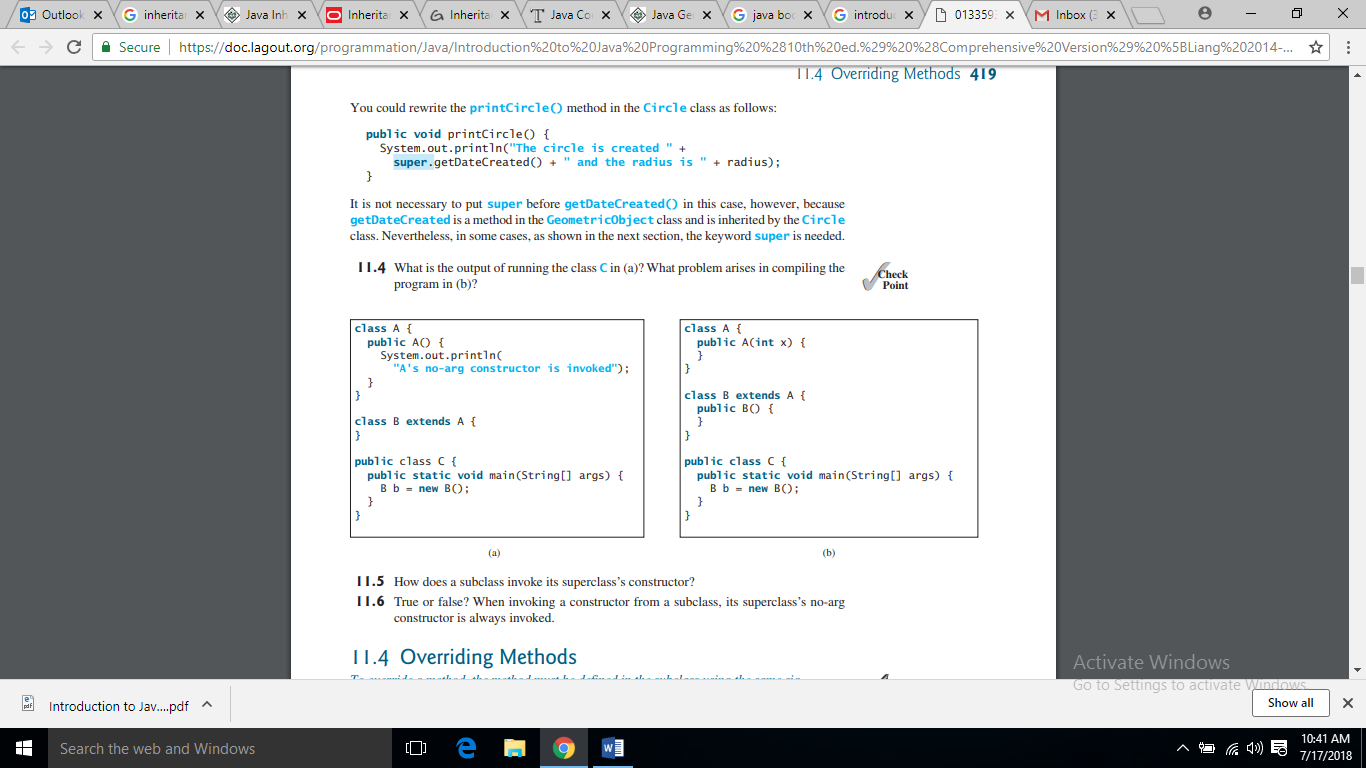
public SubClassName(double d) {

super();

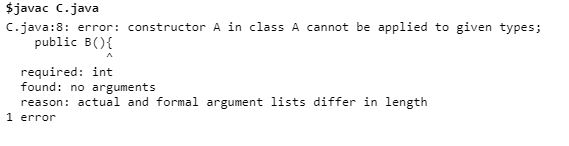
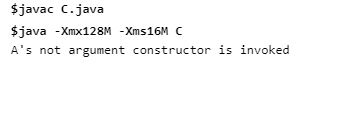
// some statements

}

In any case, constructing an instance of a class invokes the constructors of all the superclasses along the inheritance chain. When constructing an object of a subclass, the subclass constructor first invokes its superclass constructor before performing its own tasks. If the superclass is derived from another class, the superclass constructor invokes its parent-class constructor before performing its own tasks. This process continues until the last constructor along the inheritance hierarchy is called. This is called **constructor chaining**.



1. Output (b) Output

****

**Method Overloading:**

Overloading methods enables you to define the methods with the same name as long as their signatures are different.

**Why to use Method Overloading?**

Consider the following Example

class A{

public static int max(int num1, int num2) {

if (num1 > num2)

return num1;

else

return num2;

}

}

public class C

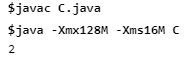
{

public static void main(String []args){

System.out.println(A.max(1,2));

}

}



The max method above works only with the int data type. But what if you need to determine which of two floating-point numbers has the maximum value? The solution is to create another method with the same name but different parameters, as shown in the following code:

public static double max(double num1, double num2)

{

if (num1 > num2)

return num1;

else

return num2;

}

If you call max with int parameters, the max method that expects int parameters will be invoked; if you call max with double parameters, the max method that expects double parameters will be invoked. This is referred to as method overloading; that is, two methods have the same name but different parameter lists within one class. The Java compiler determines which method to use based on the method signature.

**Ambiguous invocation**

Sometimes there are two or more possible matches for the invocation of a method, but the compiler cannot determine the best match. This is referred to as ambiguous invocation. Ambiguous invocation causes a compile error.

**Example:**

public class AmbiguousOverloading

{

public static void main(String[] args)

{

System.out.println(max(1, 2));

}

public static double max(int num1, double num2)

{

if (num1 > num2)

return num1;

else

return num2;

}

public static double max(double num1, int num2)

{

if (num1 > num2)

return num1;

else

return num2;

}

}

Both max(int, double) and max(double, int) are possible candidates to match max(1, 2). Because neither is better than the other, the invocation is ambiguous, resulting in a compile error.

**What is wrong in the following program?**

public class Test {

public static void method(int x){

}

public static int method(int y) {

return y;

}

}

Given two method definitions

public static double m(double x, double y)

public static double m(int x, double y)

tell which of the two methods is invoked for:

a. double z = m(4, 5);

b. double z = m(4, 5.4);

c. double z = m(4.5, 5.4);

**Method overriding:**

A subclass inherits methods from a superclass. Sometimes it is necessary for the subclass to modify the implementation of a method defined in the superclass. This is referred to as method overriding.

**Example:**

public class SimpleGeometricObject {

private String color = "white";

private boolean filled;

private java.util.Date dateCreated;

public String toString() {

return "created on " + dateCreated + "\ncolor: " + color + " and filled: " + filled;

}

To override a method, the method must be defined in the subclass using the same signature and the same return type as in its superclass.

public class CircleFromSimpleGeometricObject extends SimpleGeometricObject

{ // Override the toString method defined in the superclass

public String toString() {

return super.toString() + "\nradius is " + radius;

}

}

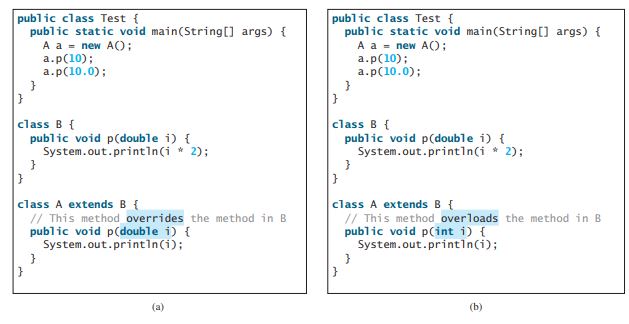
The toString() method is defined in the GeometricObject class and modified in the Circle class. Both methods can be used in the Circle class.

An instance method can be overridden only if it is accessible. Thus, a private method cannot be overridden because it is not accessible outside its own class. If a method defined in a subclass is private in its superclass, the two methods are completely unrelated.

Like an instance method, a static method can be inherited. However, a static method cannot be overridden. If a static method defined in the superclass is redefined in a subclass, the method defined in the superclass is hidden.

The hidden static methods can be invoked using the syntax SuperClassName.staticMethodName.

**Overriding vs. Overloading**



**Static Variables:**

A static variable is shared by all objects of the class. If you want all the instances of a class to share data, use static variables, also known as class variables. Static variables store values for the variables in a common memory location. Because of this common location, if one object changes the value of a static variable, all objects of the same class are affected.

Syntax: static int numberOfObjects;

**Example:**

**public class VariableDemo**

**{**

**static int count=2;**

**public void display()**

**{**

**System.out.println(count);**

**}**

**public static void main(String args[])**

**{**

**VariableDemo obj1=new VariableDemo();**

**VariableDemo obj2=new VariableDemo();**

**obj1.display();**

**obj2.display();**

**System.out.println("Obj1: count is="+obj1.count);**

**System.out.println("Obj2: count is="+obj2.count);**

**}**

**}**

**Static Methods:**

Static methods can be called without creating an instance of the class.

Syntax: static int getNumberObjects() {

return numberOfObjects;

}

**Example:**

class A{

public static int max(int num1, int num2) {

if (num1 > num2)

return num1;

else

return num2;

}

}

public class C

{

public static void main(String []args){

System.out.println(A.max(1,2));

}

}

**Instance Variables vs. Static Variables:**

Instance variables belong to the instances and have memory storage independent of one another. Static variables are shared by all the instances of the same class.

**Practice Tasks**

**Note: Draw UML class hierarchy diagram for all tasks.**

**Task 1**

Suppose you have designed a class Person that has a name, address, phoneNumber and emailAddress. Now you have to design a class Student that has a name, address, phoneNumber, emailAddress, an enrollmentId and a gpa. You have to design another class Faculty that has a name, address, phoneNumber, emailAddress, empId, salary, office hours and designation.

Which object oriented principle would you use to design these classes to avoid redundancy and provide reusability and how?

Create all classes, then create constructors, accessors and mutators for all fields and Override the **toString()** methods to display the class name and person’s name. Also override the **displayData()** methods.

Draw the UML diagram for classes and implement them. Write a test program that creates a Person, Student and Faculty and invokes their displayData() and toString() methods.

**Task 2**

IAL Saatchi & Saatchi (Pakistani large advertising firm) have approached the students of MAJU (especially the OOP students) to design and create the future episodes of Commander Safeguard animated series. They have given you general information about some characters. Now you have a very little time to understand the characters properly and design a system which clearly represents the nature of many characters.

The information from IAL Saatchi & Saatchi is as below:

*We have many characters in our series.*

*Each character has a name.*

*Every character can move and can speak.*

*Germs are one kind of character in our series.*

*The difference is that every germ has a disease.*

*The germ can fire a disease and/or can spread the disease.*

*There are some special germs that can fly.*

*The characters other than Germs are human.*

*Every human can wash hands.*

*There are basically two types of Humans; Protectors*

*Protector includes Commander Safeguard along with his team that may contain some special kids and doctors.*

Using information given above design some classes using the concept of inheritance to get the contract of being the animator of world known animated series.

**Task 3**

**(The Fan class)**

Design a class named Fan to represent a fan. The class contains:

■ Three constants named SLOW, MEDIUM, and FAST with the values 1, 2, and 3 to denote the fan speed.

■ A private int data field named speed that specifies the speed of the fan (the default is SLOW).

■ A private boolean data field named on that specifies whether the fan is on (the default is false).

■ A private double data field named radius that specifies the radius of the fan (the default is 5).

■ A string data field named color that specifies the color of the fan (the default is blue).

■ The accessor and mutator methods for all four data fields.

■ A no-arg constructor that creates a default fan.

■ A method named toString() that returns a string description for the fan. If the fan is on, the method returns the fan speed, color, and radius in one combined string. If the fan is not on, the method returns the fan color and radius along with the string “fan is off” in one combined string.

Draw the UML diagram for the class and then implement the class. Write a test program that creates two Fan objects. Assign maximum speed, radius 10, color yellow, and turn it on to the first object. Assign medium speed, radius 5, color blue, and turn it off to the second object. Display the objects by invoking their toString method.

**Task 4:**

Write a program that creates four methods. The first finds the maximum integer, the second finds the maximum double, and the third finds the maximum among three double values and fourth finds the maximum among three integer values. All four methods are named max.

**Home Task:**

1. (**Use the GregorianCalendar class)** Java API has the GregorianCalendar class in the java.util package, which you can use to obtain the year, month, and day of a date. The no-arg constructor constructs an instance for the current date, and the methods get(GregorianCalendar.YEAR), get(GregorianCalendar.MONTH), and get(GregorianCalendar.DAY\_OF\_MONTH) return the year, month, and day. Write a program to perform two tasks:

■ Display the current year, month, and day.

■ The GregorianCalendar class has the setTimeInMillis(long), which can be used to set a specified elapsed time since January 1, 1970. Set the value to 1234567898765L and display the year, month, and day.