**Administrative**

**Today’s session**

Abstract method

Abstract class

Method overriding summary

Homework 5

**Session Topics**

**Abstract method**

● The body of a **concrete method** contains one or more statements enclosed in curly braces. It has an **implementation**.

● Concrete method example:

public void printArray(String heading, int[] arr)

{

System.out.println("\n" + heading + " values");

System.out.println("Array size: " + arr.length);

System.out.printf("%-6s%6s%n", "Index", "Value");

for (int i = 0; i < arr.length; i++)

System.out.printf("%-6d%6d%n", i, arr[i]);

}

● The body of an **abstract method** is empty – there are no statements and no curly braces. It has no **implementation**.

● A method with no implementation must be declared with keywords **public** and **abstract**.

● Abstract method example:

public abstract void printArray(String heading, int[] arr);

**Abstract class**

● A **concrete class** contains no abstract methods and is not declared with keyword **abstract**.

● An **abstract class** is declared with keyword **abstract**.

● An abstract class may or may not include abstract methods. If a class has any abstract methods, it must be declared abstract.

● Abstract class example:

public abstract class ArrayTools

{

private static void randomizeArray(int[] arr, int upperLimit)

{

Random rand = new Random();

for (int i = 0; i < arr.length; i++)

arr[i] = rand.nextInt(upperLimit) + 1;

}

private abstract void printArray(String heading, int[] arr);

}

● An abstract class cannot be instantiated but can be subclassed.

● An abstract class provides a framework for subclasses to complete.

● An abstract class is considered *incomplete* but its subclasses *complete* it.

● When an abstract class is subclassed, the subclass can define implementations for all of the abstract methods in the superclass. If it does not, then the subclass must also be declared abstract.

● Abstract class and concrete subclass example:

public abstract class Employee

{

private String name;

private String hireDate;

public Employee(String theName, String theDate)

{

if (theName == null || theDate == null)

System.out.println("Error creating employee.");

name = theName;

hireDate = theDate;

}

public String getName()

{

return name;

}

public String getHireDate()

{

return new hireDate;

}

public abstract double getPay();

public void setName(String newName)

{

if (newName == null)

System.out.println("Error setting employee name.");

else

name = newName;

}

public void setHireDate(String newDate)

{

if (newDate == null)

System.out.println("Error setting employee hire date.");

else

hireDate = newDate;

}

public boolean samePay(Employee other)

{

return (this.getPay( ) == other.getPay( ));

}

public String toString( )

{

return (name + " " + hireDate.toString( ));

}

public boolean equals(Object otherObject)

{

Employee otherEmployee = (Employee)otherObject;

return (name.equals(otherEmployee.name)

&& hireDate.equals(otherEmployee.hireDate));

}

}

public class HourlyEmployee extends Employee

{

private double wageRate;

private double hours; //for the month

public HourlyEmployee(

String theName, String theDate, double theWageRate, double theHours)

{

super(theName, theDate);

if ((theWageRate >= 0) && (theHours >= 0))

{

wageRate = theWageRate;

hours = theHours;

}

else

System.out.println(Error: creating an illegal hourly employee.");

}

public double getRate( )

{

return wageRate;

}

public double getHours( )

{

return hours;

}

public double getPay( )

{

return wageRate \* hours;

}

public void setHours(double hoursWorked)

{

if (hoursWorked >= 0)

hours = hoursWorked;

else

System.out.println("Fatal Error: Negative hours worked.");

}

public void setRate(double newWageRate)

{

if (newWageRate >= 0)

wageRate = newWageRate;

else

System.out.println("Fatal Error: Negative wage rate.");

}

public String toString( )

{

return (getName( ) + " " + getHireDate( ) + "\n$" + wageRate +

" per hour for " + hours + " hours");

}

public boolean equals(HourlyEmployee other)

{

return (getName( ).equals(other.getName( )) &&

getHireDate().equals(other.getHireDate( )) &&

wageRate == other.wageRate && hours == other.hours);

}

}

● See **Abstract class** sample application on Blackboard.

**Method overriding summary**

● Method overriding only applies to inherited methods.

● The overriding method must have the same name and parameter list as the overridden method (otherwise its overloading).

● The overriding method must have the same return type, or a subclass of the return type.

● The overriding method may not have a more restrictive access modifier, but may have a less restrictive one

● Constructors cannot be overridden.

● Final methods cannot be overridden.

● Static methods can be overridden but become hidden yet remain accessible with superclass name prefix.

● Abstract methods must be overridden.

● The class used to create an object determines which overridden method will be called at runtime (this is polymorphism).

**Homework 5**

● Assigned today.

● Available on Blackboard.

● Due in one week.