**Exercises – Super and Subclass** Name: Taylan Unal

Period: 6

1) Define a class for a Vehicle with the following properties:

Data fields: name, # passengers

Methods: 2 argument constructor, toString, getName(), getPassengers(), setName()

**public** **class** Vehicle {

**private** String name;

**private** **int** pass;

**public** Vehicle(){

String name="Toyota";

**int** pass=1;

}

**public** Vehicle(String n, **int** p){

String name = n;

**int** pass = p;

}

**public** String toString(){

**return** "The car was made by " + name + " and it holds " + pass + " passengers";

}

**public** String getName(){

**return** name;

}

**public** **int** getPassengers(){

**return** pass;

}

**public** **void** setName(String newn){

name=newn;

}

}

2) Define a subclass of Vehicle called Car with the following properties:

Data fields: odometer value

Methods: 3 argument constructor, toString (overloaded to also show odometer),

getOdometer(), and driveAMile (advances odometer value by 1)

**public** **class** Car **extends** Vehicle{

**private** **double** odomval;

**public** Car(){

**double** odomval = 42.1;

}

**public** String toString(){

**return** "Your odometer reads " + odomval;

}

**public** **double** getOdom(){

**return** odomval;

}

**public** **void** driveAMile(){

odomval++;

}

}3) Describe a logical class definition and multiple subclass structure of your own design. A super class will have a subclass, which in turn will also have a subclass of its own. You do not need to write the code. Only describe the following:

**Super class: what data fields and methods would you have?**

Class: Communication

Boolean transfer

String sender

String reciever

**Subclass 1 extends Super class: what additional data fields and methods?**

Class: Cell Phone

String network technology

Boolean mobile

**Subclass 2 extends Subclass 1: what additional data fields and methods?**

Class: Galaxy S6

String screentype

String resolution

Double price

String carrier

**Exercises – Abstract Class** Name: Taylan Unal

Period: 6

Consider the following class:

//in a file called Organism.java

**public abstract class Organism**

**{**

**private double weight;** //data fields

**private String name;**

**public Organism(double w, String n)**

**{**

**weight = w;**

**name = n;**

**}**

**public double getWeight()**

**{**

**return weight;**

**}**

**public void setWeight(double w)**

**{**

**weight = w;**

**}**

**public abstract void grow();** //abstract methods are not defined

// but are overloaded in sub-classes

**public String toString()**

**{**

**return "The organism weighs " + weight + " lbs";**

**}**

**}**

1) Define a sub-classes of Organism that will simulate a mammal that will gain 2.5% of its weight when it grows (provided that its weight is under 120).

2) Define your own abstract class and two subclasses. For it, consider the following: what is a method that every subclass should have defined concretely, and differently from one another that is too ambiguous to define for the base class? That will be your abstract method in the base class.

Exercises **Interfaces** Name\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Period\_\_\_\_\_\_\_

In your own words, answer the following questions:

1) What are the implications of deriving a subclass from an abstract class?

2) What are the implications of having a class implement an interface?

3) What is the difference between an abstract class and an interface?

Consider the following definitions:

public abstract class *Collector* extends *Robot* public interface *Jumpable*

{ {

public Collector() public abstract void jumpUp();

{ public abstract void jumpDown();

super(); public abstract void jumpSide();

} }

public void getBeepers()

{

while(onABeeper())

pickBeeper();

}

public abstract void collectAll();

}

You make a subclass of Collector like this: public class *Grabber* extends *Collector*.

4) List the names of the methods do you automatically inherit (including constuctors)?

5) List the names of the methods **must** you define in *Grabber*(including constuctors)?

You make a new robot class like this: public class *Hopper* implements *Jumpable.*

6) List the names of the methods do you automatically inherit(including constuctors)?

7) List the names of the methods **must** you define in *Hopper*(including constuctors)?

You redefine *Hopper* like this:

public class *Hopper* extends *Collector* implements *Jumpable.*

8) List the names of the methods do you automatically inherit(including constuctors)?

9) List the names of the methods **must** you define in *Hopper*(including constuctors)?

**public** **interface** ShapeMath //in a file called shapeMath.java

{

//post: returns the area of a specific type of geometric shape

**double** findArea();

//post: returns the circumference/perimeter of a shape

**double** findCircumference();

}

10) Define two classes that implement the ShapeMath interface. One class will define a Rectangle, the other will define a Circle. Include any necessary data fields for each, and any required methods.

**public class** Rectangle **implements** ShapeMath

{ //define appropriate data field(s), constructor & method(s)

}

**public class** Circle **implements** ShapeMath

{ //define appropriate data field(s), constructor & method(s)

}

Exercises **Pre and Post Conditions** Name\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Period\_\_\_\_\_\_\_

Give the proper pre- and post-conditions for the following pseudo-code methods. Make sure that you actually LOOK closely at the method, and perhaps step trace through it with some values. For the pre-condition, consider what the input must be for the method to do what is expected and in the correct way. For the post-condition, consider what the method does.

//pre:

//post:

public static void no1(int[] nums, int a, int b)  
{  
 int temp = nums[a];  
 nums[a] = nums[b];  
 nums[b] = temp;  
}

//pre:

//post:

public static int no2(int[] nums)  
{  
 if (nums == null)  
 return Integer.MIN\_VALUE;  
 int max = Integer.MIN\_VALUE;  
 for (int i=0; i < nums.length; i++)  
 if (nums[i] > max)  
 max = nums[i];  
 return max;  
}

//pre:

//post:

public static int[] no3(int[]array, int scalar)  
{  
 int[] retVal = array.clone(); //makes *retVal* a clone of *array* for(int i=0; i<retVal.length; i++)  
 {  
 if(retVal[i] >= 0)  
 retVal[i] += scalar;  
 }  
 return retVal;  
}

//pre:

//post:

public static int no4(int ourNote, int[] scale)  
{  
 for(int i=0; i<scale.length; i++)  
 if(ourNote == scale[i])  
 return i;  
 return -1;  
}

//pre:

//post:

public static int no5(int ourNote)  
{  
 while(ourNote>=12)  
 ourNote-=12;   
 return ourNote;  
}

Note: the for:each loop is an easy way to traverse through every element of an array.

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

System.out.println(array[i]);

can be done as:

for(int num : array)

System.out.println(num);

//pre:

//post:

public static int[] no6(int[]a, int[]b)  
{  
 int[]retVal = new int[a.length + b.length];  
 int index = 0;  
 for(int num:a) //*num* will traverse thru every element of *a*

{  
 retVal[index] = num;

index++;

}  
 for(int num:b) //*num* will traverse thru every element of *b*  
 retVal[index++] = num; //same as first loop, but for array *b*  
 return retVal;  
}