10/30/15 APCS – Monta Vista HS Classes, Strings, Hierarchy Name_____

1) Determine what will be printed by the following code fragment.

Printed:

```
String word = new String("JUMBLE");
String result = new String("");
for(int i = 1; i < word.length(); i++)
{
        result += word.substring(i,i+1);
        if(i+2 < word.length())
        {
                result += word.substring(i,i+2);
        }
        System.out.println(result);
}
```

2) Determine what will be returned for the following method definition and method call.

guessTheString("MickeyMouse")

```
public String guessTheString(String word) {
        String result = new String("");
        char first = word.toLowerCase().charAt(0);
        switch (first) {
                case 'a': case 'e': case 'i': case 'o': case 'u':
                        result = word + "-ay";
                        break;
                default:
                        result = word.substring(1,2).toUpperCase() +
                                word.substring(2) + "-" + first + "ay";
        }
        return result;
}
```

3) Suppose that class **Sporty** is derived from the class **Vehicle**. Consider the following statements:

```
Vehicle  redCar = new Vehicle();
Vehicle  blueCar;
Sporty  s2000 = new Sporty();
Sporty  rx7;
blueCar = s2000;
```

Which of the following statements is/are legal in Java? Circle all that apply.

(a) **rx7 = (Sporty) redCar;**

(b) **rx7 = (Sporty) blueCar;**

(c) **rx7 = (Sporty) s2000;**

Use the following code to answer questions 4 and 5. You have the following classes:

```
class MomClass {
        public static void staticMethod() {
                System.out.println("MomClass staticMethod");
        }
        public void instanceMethod() {
                System.out.println("MomClass instanceMethod");
        }
}

class SonClass extends MomClass {
        public static void staticMethod() {
                System.out.println("SonClass staticMethod");
        }
        public void instanceMethod() {
                System.out.println("SonClass instanceMethod");
        }
}
```

The program creates two objects from these classes.

```
SonClass sc = new SonClass();
MomClass mc = new SonClass();
```

4) Write what is printed by the next two lines.

```
sc.instanceMethod();
mc.instanceMethod();
```

5) Write what is printed by the next two lines.

```
sc.staticMethod();
mc.staticMethod();
```

6) Suppose a project needs two classes, **Predator** and **Prey**; each class has the methods **breed**, **age**, and **hunt**. The code for **breed** is identical for **Predator** and **Prey**. The code for **age** is very close for **Predator** and **Prey**: it starts the same way but has differences at the end. The code for **hunt** is totally different for **Predator** and **Prey**. Which of the following would be an appropriate design in this situation?

(A) Define an interface **Animal** with **breed**, **age**, and **hunt** methods; make **Predator** and **Prey** implement **Animal** by supplying appropriate code for the methods.
(B) Make **Predator** a superclass and **Prey** its subclass; override **age** and **hunt** in **Prey**.
(C) Define a class **Animal** with a method **breed**; derive both **Predator** and **Prey** from it, adding **age** and **hunt** methods in each of them.
(D) Define **Predator** and **Prey** separately, defining methods **breed**, **age** and **hunt** within each class.
(E) Define an abstract class **Animal** with methods **breed** and **age** (placing there only the part of the code that is common for **Predator** and **Prey**) and an abstract method **hunt**; derive both **Predator** and **Prey** from this abstract class, supplying the rest of the code for **age** and the code for **hunt** in each of them.

7) Consider the following classes:

```
public class Horse {
        private String name;

        public Horse(String n) {
                name = n;
        }

        public void setName(String n) {
                name = n;
        }

        public String getName() {
                return name;
        }
}

public class Arabian extends Horse {
        private String prize;
        public Arabian(String n, String p) {

                super(n);                    // LINE 1
                prize = p;
        }
        public String getPrize() {
                return prize;
        }
}
```

Which of the following can replace **LINE 1** in the **Arabian** class and perform the same operation?

(a) **name = n;**

(b) **setName(n);**

(c) **super.setName(n);**

(d) **both (b) and (c)**

(e) **None of the above will work**

8) (**6 points**) Given the following local variable declarations, find the results of the following expressions (or ERROR).

```
String goodGuy = new String("road runner");
String badGuy = new String("coyote");
String oneChar = "A";
String animal = goodGuy;
goodGuy = "road runner";
```

Expression	Return Value	Data Type
badGuy.length()		
badGuy.indexOf('e')		
oneChar + 5		
animal == goodGuy		
animal.equals(goodGuy)		
badGuy.substring(0,3) + goodGuy.substring(4)		

9) Write a method **exOut** that receives two Strings in its parameter list, **phrase** and **keep**. The method **exOut** should return a version of the **String phrase** where all chars have been replaced by a capital X (**"X"**), except for appearances of the **String keep**, which are preserved unchanged. You are not allowed to use the methods **replace** or **replaceAll** from the java **String** class. See the examples below.

exOut ("Good work", "o") returns **"XooXXXoXX"** **exOut ("Hello there!", "ere")** returns **"XXXXXXXXereX"**
exOut ("This is bliss.", "is") returns **"XXisXisXXXisXX"**

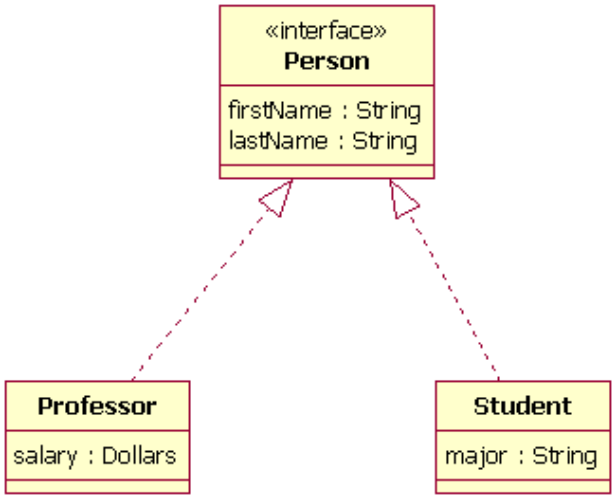
public String exOut (String phrase, String keep)

10) Write the method **repeatBuild** that receives one **String** in its parameter list, **word**. The method **repeatBuild** should create a new **String** from the **String word**. This new **String** should be created by adding the **word** repeatedly, removing a letter from the end each time until there is nothing left to add. Do not solve this problem by using recursion. See the examples below:

repeatBuild("Monta"); returns **"MontaMontMonMoM"** **repeatBuild("cool");** returns **"coolcoococ"**
repeatBuild("Crazy"); returns **"CrazyCrazCraCrC"** **repeatBuild("Loopy");** returns **"LoopyLoopLooLoL"**
repeatBuild(""); returns **""**

public String repeatBuild(String word)

11) (4 points) Consider the following inheritance hierarchy diagram:

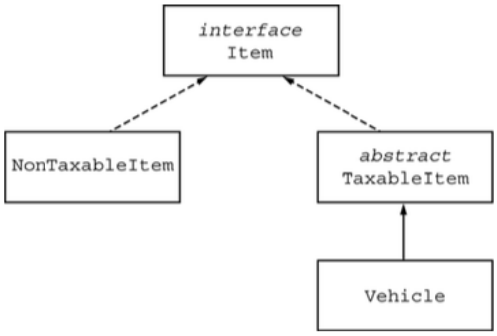


Which of the following declarations will **not** cause an error? You may assume that each of the classes has a no-args constructor.

(i) **Person lorde = new Professor();**
(ii) **Student taylor = new Professor();**
(iii) **Person mariah = new Student();**
(iv) **Person gaga = new Person();**

(A) none of these
(B) (i), (ii), and (iii) only
(C) (ii), (iii), and (iv) only
(D) (i) and (iii) only
(E) (i), (ii), (iii), and (iv)

12) A set of classes is used to represent various items that are available for purchase. Part of the class hierarchy is shown in the diagram below.



The definitions of the **Item** interface and the **TaxableItem** classes are shown below.

```
public interface Item
{
        double purchasePrice();
}

public abstract class TaxableItem implements Item
{
        private double taxRate;
        public abstract double getListPrice();

        public TaxableItem(double rate)
        {       taxRate = rate;         }

        public double purchasePrice()           // returns the price of the item including the tax
        {
                /* to be implemented by you */
        }
}
```

(a) (5 points) Write the **TaxableItem** method **purchasePrice**. The purchase price of a **TaxableItem** is the list price plus the tax on the item. The tax is computed by multiplying the list price by the tax rate. For example, if the tax rate is 0.10 (10%), the purchase price of an item with list price of $6.50 would be $7.15.

Complete the method **purchasePrice** below.

```
// returns the price of the item including the tax
public double purchasePrice()
```

(b) (15 points) Create the **Vehicle** class, which extends the **TaxableItem** class. A **Vehicle** has two parts to its list price: a dealer cost and dealer markup. The list price of a **Vehicle** is the sum of the dealer cost and the dealer markup.

For example, if a vehicle has a dealer cost of $20,000.00, a dealer markup of $2,500.00, and a tax rate of 0.10, then the list price of the vehicle would be $22,500.00 and the purchase price (including tax) would be $24,750.00.

Your class should have a constructor that takes dealer cost, the dealer markup, and the tax rate as parameters (all doubles). Provide any private instance variables needed and implement all necessary methods. Also, provide a public method **changeMarkup**, which changes the dealer markup to the value of its parameter.

13) (8 points) Determine what will be printed from the following program.

Printed:

```
class Citizen {
        public static void speech() {
                System.out.println("off the cuff ");
        }
        public void handshake() {
                System.out.println("shake on it ");
        }
}

class Candidate extends Citizen {
        public static void speech() {
                System.out.println("make promise ");
        }
        public void handshake() {
                System.out.println("many times ");
        }
}

public class MemberOfCongress extends Candidate {
        public static void speech() {
                System.out.println("canned ");
        }
        public void handshake() {
                System.out.println("sincere ");
        }

        public static void main(String[] args) {
                Citizen chu = new Candidate();
                chu.handshake();
                chu.speech();

                Citizen gopal = new MemberOfCongress();
                gopal.handshake();
                gopal.speech();

                MemberOfCongress jones = new MemberOfCongress();
                jones.handshake();
                jones.speech();

                Candidate smith = jones;
                smith.handshake();
                smith.speech();
        }
}
```

14) (8 points) Determine what will be printed from the following program.

```
public class WorkingWithParameters
{
	public static void main (String [] args)
	{
		WorkingWithParameters run = new WorkingWithParameters();
		run.changeValues();
		run.changeMore();
	}
	public void changeValues ( )
	{
		char letter1 = 'c', letter2 = 's';
		printLetters(letter1, letter2);
		changeLetters(letter1, letter2);
		printLetters(letter1, letter2);
	}

	public void printLetters (char let1, char let2)
	{
		System.out.println(let1 + "  " + let2);
	}

	public void changeLetters (char let1, char let2)
	{
		let1 = 'a';
		let2 = 'p';
		printLetters(let1, let2);
	}

	public void changeMore ( )
	{
		String course = "APCS", name = "JAVA";
		printStrings(course, name);
		changeStrings(course, name);
		printStrings(course, name);
	}

	public void printStrings (String s1, String s2)
	{
		System.out.println(s1 + "  " + s2);
	}

	public void changeStrings (String s1, String s2)
	{
		s1 = new String("MONTA");
		s2 = new String("VISTA");
		printStrings(s1, s2);
	}
}
```

15) (8 points) Determine what will be printed from the following program.

```
public class MoreParameters2
{
	public static void main (String [] args)
	{
		MoreParameters2 run = new MoreParameters2();
		run.changeArray();
	}

	public void changeArray ( )
	{
		int [] array = {8, 33, 52};
		printArray(array);
		changeArrayOne(array);
		printArray(array);
		changeArrayTwo(array);
		printArray(array);
	}

	public void printArray (int [] arr)
	{
		System.out.println(arr[0] + "  " + arr[1] + "  " + arr[2]);
	}

	public void changeArrayOne (int [] arr)
	{
		arr[1] = arr[2];
		arr[2] = arr[0];
		arr[0] = arr[1];
		printArray(arr);
	}

	public void changeArrayTwo (int [] arr)
	{
		arr = new int[3];
		arr[0] = 9;
		arr[1] = 13;
		arr[2] = 17;
		printArray(arr);
	}
}
```