Beijing National Day School
Department of Mathematics

AP Computer Science A

Test 3: `Inheritance`

Exam Record	
Multiple Choice	/ 19 pts
Java Programs	/ 27 pts
Total:	/ 46 pts
Grade:	

English Name: ________________

Pinyin Name: ________________

Mr. Alwin Tareen, Spring 2019

Section I: Multiple Choice (19 points)

- Decide which is the best of the choices given, and select the correct answer by placing an "**X**" in the corresponding box.

(1^{pt}) **1.** What is the keyword that a subclass uses to indicate that it is inheriting from a particular superclass?

☐ `extends`
☐ `diverges`
☐ `includes`
☐ `implements`

1 pt

(1^{pt}) **2.** What is the keyword that a subclass uses to indicate that it is using a superclass that has been declared as an interface?

☐ `extends`
☐ `diverges`
☐ `includes`
☐ `implements`

1 pt

(1^{pt}) **3.** The process of inheritance refers to the fact that a subclass inherits some of the characteristics from its:

☐ sibling class
☐ reference class
☐ superclass
☐ primal class

1 pt

(1^{pt}) **4.** What restriction is there on using the `super` reference in a constructor?

☐ It can only be used in the parent's constructor.
☐ Only one child class can ever use it.
☐ It must be used in the last statement of the constructor.
☐ It must be used in the first statement of the constructor.

1 pt

(1^{pt}) **5.** Which of the following choices is one of the characteristics of an `abstract` class?

☐ An `abstract` class is one without any subclasses.
☐ An `abstract` class is any superclass with more than one subclass.
☐ An `abstract` class is a class which cannot be instantiated.
☐ An `abstract` class is another name for "superclass".

1 pt

(1^{pt}) **6.** Consider a superclass `Video` that has subclasses `Movie` and `MusicVideo`. Select the statement that will compile without error.

☐ `Movie frozen = new MusicVideo();`
☐ `MusicVideo adele = new Video();`
☐ `Movie titanic = new Video();`
☐ `Video avatar = new Movie();`

1 pt

6 pts

(1^{pt}) **7.** What is an `abstract` method? | 1 pt

- ☐ An `abstract` method is any method in an abstract class.
- ☐ An `abstract` method is a method that cannot be inherited.
- ☐ An `abstract` method is a method header without a body.
- ☐ An `abstract` method is a method in a subclass that overrides a method in the superclass.

(1^{pt}) **8.** Can an `abstract` class define *both* `abstract` methods and non-`abstract` methods? | 1 pt

- ☐ No, it must have all of one, or all of the other.
- ☐ No, it must have all `abstract` methods.
- ☐ Yes, but the subclasses do not inherit the `abstract` methods.
- ☐ Yes, and the subclasses inherit both types of methods, if they are `public`.

(1^{pt}) **9.** What must be true if a subclass of an `abstract` superclass does *not* override all of the `abstract` methods from that superclass? | 1 pt

- ☐ Inheritance would not be possible in this case, so the subclass must be discarded.
- ☐ The subclass itself must be declared as `abstract`.
- ☐ Subclasses are automatically declared as non-`abstract`, so there would be no error.
- ☐ The superclass would cause a compiler error.

(1^{pt}) **10.** What is the term for the restriction whereby a subclass `extends` from a single superclass? | 1 pt

- ☐ single inheritance
- ☐ multiple inheritance
- ☐ compiler inheritance
- ☐ garbage collection

(1^{pt}) **11.** Can a subclass inherit constructors from its superclass? | 1 pt

- ☐ Yes, constructors are *always* inherited.
- ☐ Yes, as long as the constructors are declared `private`.
- ☐ No, because subclasses are not permitted to have any constructors.
- ☐ No, constructors are *never* inherited.

(1^{pt}) **12.** Which of the following is the best description of an interface? | 1 pt

- ☐ It is a type of class which cannot contain `abstract` methods.
- ☐ It contains a collection of related `abstract` methods.
- ☐ It is a method which is declared `private` and `abstract`.
- ☐ It is a type of constructor which gets inherited from a superclass.

(1^{pt}) **13.** What is a downcast? | 1 pt

- ☐ This is where a method is converted from `abstract` to non-`abstract`.
- ☐ Choosing which type of overridden method to run on an object.
- ☐ The case where an object's data type is changed from superclass type to subclass type.
- ☐ Conversion from an `abstract` class to an `interface`.

7 pts

(2pts) **14.** What is wrong with this interface?

2 pts

```
public interface Faulty
{
   void someMethod(String password)
   {
      System.out.println("The password is: " + password);
   }
}
```

☐ A method in an interface should be declared `public`.
☐ A method in an interface should be declared `abstract`.
☐ There should not be a method body with code inside.
☐ There should be a class implementation provided.
☐ There should not be any method parameters.

(2pts) **15.** Consider the following class declarations:

2 pts

```
public class Person
{ . . . }
public class Teacher extends Person
{ . . . }
```

Which of the following statements is *true*?

I `Teacher` inherits the constructors of `Person`.
II `Teacher` can add new methods and private instance variables.
III `Teacher` can override existing private methods of `Person`.

☐ I only
☐ II only
☐ III only
☐ I and II only
☐ II and III

(2pts) **16.** Consider the following class declarations:

2 pts

```
public class Animal
{
   public void eat()
   { /* implementation of eat */ }
}
public class Dog extends Animal
{
   public void eat()
   { /* different implementation of eat */ }
}
```

The `eat()` method in class `Dog` is an example of:

☐ method overloading.
☐ method overriding.
☐ polymorphism.
☐ information hiding.
☐ procedural abstraction.

6 pts

Section II: Java Programs (27 points)

- Show all of your work. Remember that program segments are to be written in the `Java` programming language.

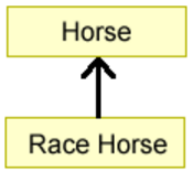
(9pts) **1.** A `Horse` class is defined with the following instance variables, constructors and methods.

9 pts

```
public class Horse
{
    // instance variables
    private String owner;
    private int age;
    private double value;

    // constructors
    public Horse()
    {
        owner = "";
        age = 0;
        value = 0;
    }

    public Horse(String own, int a, double v)
    {
        owner = own;
        age = a;
        value = v;
    }

    // accessor methods
    public String toString()
    {
        String result = "";
        result += "Owner = " + owner + "\n";
        result += "Age = " + age + "\n";
        result += "Value = $" + value;
        return result;
    }
}
```

19 pts

Define a `RaceHorse` class that inherits from the `Horse` class, but has an additional private instance variable that indicates the number of races that the horse has won. The relationship between `RaceHorse` and `Horse` can be illustrated by the following UML diagram:



Use the following guidelines to implement the `RaceHorse` class.

(a) (1 pt) Create a private instance variable of type `int` named `numRacesWon`.

(b) (2 pts) Create a default constructor which initializes the instance variables to 0.

(c) (2 pts) Create a constructor which takes 4 parameters to initialize the instance variables `owner`, `age`, `value`, and `numRacesWon`.

(d) (1 pt) Implement the accessor method `getRacesWon` which has the return type `int`.

(e) (1 pt) Implement the mutator method `wonRace` which has the return type `void`.

(f) (2 pts) Override the `toString` method so that it includes `numRacesWon`.

Write your solution for the `RaceHorse` class in the space below.
You may also use the space provided on the next page.

−10 pts

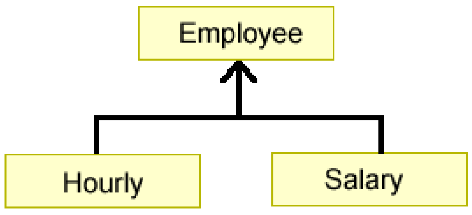
The `RaceHorse` class:

0 pts

(9pts) **2.** A company has two types of categories for the various employees that it hires. Hourly workers are assigned a wage which is paid for every hour that they work. Salary workers, on the other hand, are designated a yearly salary as their wage.

9 pts

According to company policy, each type of employee must be paid at the end of every month. Since the two types of employees require different calculations for their monthly pay, the method which performs this action, `monthlySalary()`, is provided as an `abstract` method in the `Employee` class. The relationship between the `Employee` class and the `Hourly` and `Salary` classes is shown in the following UML diagram:



The `Employee` class is specified as an `abstract` class, and is shown in the following code listing:

```
public abstract class Employee
{
    // instance variables
    private String name;
    private int IDNumber;

    // constructors
    public Employee(String n, int i)
    {
        name = n;
        IDNumber = i;
    }

    // accessor methods
    public String getName()
    {
        return name;
    }

    public int getIDNumber()
    {
        return IDNumber;
    }

    // abstract methods
    public abstract double monthlySalary();
}
```

0 pts

(a) (5 pts) Implement the `Hourly` class, using the following guidelines:

- Create two instance variables to represent the wage rate per hour, and the number of hours worked each month.
- Create a constructor which takes four parameters to initialize the employee's name, ID number, hourly wage, and hours worked for the month.
- Implement the method `monthlySalary()` which calculates the monthly salary paid to the employee. The monthly salary is calculated by multiplying the number of hours worked each month by the hourly wage rate. It should return a `double` data type.

Write your solution for the `Hourly` class in the space below:

5 pts

(b) (4 pts) Implement the `Salary` class, using the following guidelines:

- Create an instance variable to represent the yearly salary that the employee receives.
- Create a constructor which takes three parameters to initialize the employee's name, ID number, and yearly salary.
- Implement the method `monthlySalary()` which calculates the monthly salary paid to the employee. The monthly salary is calculated by dividing the yearly salary by `12.0`. It should return a `double` data type.

Write your solution for the `Salary` class in the space below:

4 pts

(9^{pts}) **3.** You are provided with the interface `LinearFunctionMethods` described below, and you are required to create a `LinearFunction` class that inherits from this interface, and provides implementations for all of the methods.

9 pts

```java
public interface LinearFunctionMethods
{
    double getSlope();
    double getYintercept();
    double getXintercept();
    double getYvalue(double x);
    double getXvalue(double y);
}
```

These methods are based on the equation of line in a 2-D plane: $y = mx + b$. You may need to perform some simple algebra on this equation to correctly implement these methods. Assume that the linear function's graph can never be vertical or horizontal. Let your class `LinearFunction` be instantiated in a test bench in the following manner:

```java
LinearFunction line = new LinearFunction(slope, yintercept);
```

(a) (2 pts) Create two instance variables to represent the line's slope and y-intercept values.

(b) (2 pts) Create a constructor that takes two parameters to initialize the instance variables.

(c) (5 pts) Provide method implementations for all of the abstract methods indicated in the `LinearFunctionMethods` interface.

Write your solution for the `LinearFunction` class in the space below.
You may also use the space provided on the next page.

9 pts

The `LinearFunction` class:

0 pts