Exam #1 Practice #1 Key

VOCAB KEY

1 Subclass

2 Abstract Method

3 Garbage Collector

4 Definition

5 Assertion

6 Abstraction

7 Declaration

8 Namespace

9 Destructor

10 Variable

MULTIPLE CHOICE KEY

1 B	6 C	11 D
2 C	7 B	12 D
3 D	8 B	13 A
4 C	9 C	14 A
5 D	10 C	15 A

Free Response

Code won't match exactly (e.g., variable names may change or a different algorithm may be used). Points rubric is in the code comments, and are a guide only. Grader discretion required.

1.a {4 points}

```
// 1 for "public", 1 for "enum Light", 2 for the rest
// -1 if values are in double quotes, no deduction for ; at the end
public enum Light {FULL_SUN, PART_SHADE, SHADE}
```

b. {7 points}

```
// 1  point for public class Plant
// 2  points for constructor
// 1  point (½ each) for name() and light()
// 1  point for @Override
// 1  point for method toString()
// 1  point (½ each) for the 2 fields

public class Plant {
  public Plant(String name, Light light) {
      this.name = name;
      this.light = light;
  }
  public String name() {return name;}
  public Light light() {return light;}
  @Override
  public String toString() {
      return name + " (" + light + ")";
  }
  private String name;
  private Light light;
}
```

c. {10 points}

```java
// Import is not required on exams

import java.util.ArrayList;

public class Garden {

   // 1 for correctly declaring the constructor
   // 1 point for correct use of this.
   // 1 points for a correct constructor implementation

   public Garden(Light light) {
       this.light = light;
       this.plants = new ArrayList<>();
   }

   //  ½ for correctly declaring the method (incl. curly braces)
   // 1½ points for adding plant to plants (missing _ OK)

   public void addPlant(Plant plant) {
       plants.add(plant);
   }


   // ½ for @Override
   // ½ for correctly declaring the method (incl. public & curly braces)
   // ½ point for a valid for each (or 3-term for) loop
   // 1 for a correct concatenation into a method-scoped String
   // ½ for a correct return

   @Override
   public String toString() {
       String result = "";
       String separator = "";
       for(Plant p : plants) {
           result += separator + p;
           separator = ", ";
       }
       return result;
   }

   // 2 (1 each) for correct ArrayList and Light declarations

   private ArrayList<Plant> plants;
   private Light light;
}
```

2.
   a. Animal cannot be instanced, because Animal.speak() is abstract and thus Animal is abstract {3 points}
   b. `@Override String speak() {return "Meow";}` {3 points}
   c. {6 at 1 point each, 6 points total}

Cat will inherit Animal's default constructor.

F - Constructors never inherit

Cat.speak() can access Animal.heart_rate.

T - Animal.heart_rate is protected, which is accessible from a derived class

Cat.speak() can access Animal.warm_blooded.

F - Animal.warm_blooded is private, which is NOT accessible from a derived class

`Cat c = new Cat(); c.breath();` will compile without errors.

T - Cat inherits Animal.breath(), so the code will compile

The relationship of Cat to Animal is usually expressed in Java as `class Cat extends Animal`;

T - That's exactly how inheritance is expressed in Java

T F The return type for Animal.breath() is undefined

F - Animal.breath() has a void return type, which is not undefined

3.

a. {3 points}

```
// 3 points - 1 point per line
interface Gradeable {
    public void grade();
    public double getGrade();
}
```

b. {4 points}

```
// 1½ "implements Gradeable"
class Exam implements Gradeable {

    // 1 point @Override
    // 1 point (½ per method)
    @Override
    public void grade() {
        score = 40 + 60 * Math.random();
    }
    public double getGrade() {
        return score;
    }
    // ½ point for field
    private double score;
}
```

4. {10 points}

```
import java.util.Scanner;

public class Bank {
    public void deposit(double amount) {
        // Add code so that, if amount is not positive,
        //   an Illegal Argument Exception is thrown
        //   with the message "Non-positive deposit amount"

        // 1 for the if conditional (with or without { }
        // 1 for throw
        // 1 for new
        // 1 for IllegalArgumentException
        // 1 for the message as a parameter

        if (amount <= 0)
           throw new IllegalArgumentException("Non-positive deposit amount");

        balance += amount;
   }

  public static void main(String[] args) {

      // Add code so that, if the user enters a non-positive
      //   deposit amount below, catch the exception
      //   thrown in Bank.deposit(double) and print
      //   its message to the console's error output stream

       Bank bank = new Bank();
       Scanner in = new Scanner(System.in);
       double d = in.nextDouble();

       // 1 for the try clause
       // 1 for the catch clause
       // 1 for the catch parameter (Exception also OK)
       // 2 for the catch body (1 for System.err, 1 for e.getMessage() or e)

       try {
           bank.deposit(d);
       } catch (IllegalArgumentException e) {
           System.err.println(e.getMessage());
       }
    }

    private int balance;
}
```

Bonus: {4 points}

1 point for each the license name and 1 point for each description

Options include:

- Public Domain for which all ownership is disclaimed (SQLite)
- Permissive (MIT, BSD, Apache) permits use, copying, distribution, and (usually with attribution) derivatives
- Protective (GPL 2, 3, Lesser GPL, EPL) permits use, copying, distribution, and derivatives with share-alike rules
- Shareware permits (sometimes limited) use, copying, and (usually) distribution
- Proprietary permits (often restricted) use according to an End User License Agreement (EULA)
- Trade Secret typically restricts use to the copyright holder