Exam A – Solutions

Exam B has the same multiple choice questions, but in different order.

2. Which *one* of the following is used in UML diagrams to signal that a *member* is abstract?(a) underline (b) *italics* (c) **boldface** (d) CAPITALIZE (e) none of the above

3. **True** (a) or **False** (b): If a class is abstract, then all of its members must be abstract

4. Which *one* of the following methods fulfills the role of `main()` in a JavaFX application?
(a)`init()` (b) `start()` (c) `launch()` (d) `stage()` (e) none of the above

5. Circle the *single best answer*: The definition

```
ArrayList<String> months = new ArrayList<>();
```

(a) instantiates an ArrayList object named `months` that is no different from an array defined as

```
String[] months = new String[];
```

(b) flags an error, since the generic type is unspecified on the right hand side of the definition

(c) instantiates a special kind of array, one that initially holds a single String element, by default

(d) uses the identifier `months` to hold an ID that points to a simple array of Strings

(e) none of the above

Note: the size of the arraylist is initially 0; hence it does not hold *any* strings at startup

6. **TRUE** (a) or **FALSE** (b): public static members in a superclass are not visible in any of its subclasses

7. **TRUE** (a) or **FALSE** (b): *every* class must be declared in a file that has the same name as the class itself

Note: inner classes do not need to have the same name as the file in which they are found.

8. When `public static void main(String[] args)` is written in a UML class diagram, it will appear as:

```
(a)+main(args: String[]): void
(b)–main(args: String[]): void
(c)+main(args: String[]): void
(d)+main(): void
(e)none of the above
```

9. Which *one* of the following is **NOT TRUE** of a class that contains an abstract method:

(a) The class cannot be extended to a subclass
(b) The class cannot be instantiated
(c) The abstract method must be overridden in a subclass, unless the subclass is itself abstract
(d) The class can be extended into a concrete class
(e) None of the above

10. Which one of the following symbols would be used to indicate that one class has many other classes (i.e. a '*has a*' relationship involving one-to-many classes)?

(a)  (b)  (c)  (d)  (e) none of the above

11. **TRUE** (a) or **FALSE** (b): ArrayList<> can only be used to store reference types; you cannot store primitive types in an ArrayList directly with their being converted to their class equivalent type.

12. Which abstract JavaFX class is the parent superclass of all graphical JavaFX objects?
(a) Node (b) Control (c) Pane (d) Parent (e) none of the above

13. **TRUE** (a) or **FALSE** (b): Lambda expressions can only be used with classes that contain a single method only.

14. **TRUE** (a) or **FALSE** (b): The `new` keyword is implied whenever you make a definition like:

```
String month = "December";
```

15. **TRUE** (a) or **FALSE** (b): You can always extend any superclass into a subclass; there are no restrictions

16. Which *one* of the following is ***NOT*** one of the three basic reference types in Java?
(a) a String (b) a class (c) an interface (d) an array (e) none of the above

17. Which *one* of the following classes contains static methods designed to allow the programmer to sort through a simple Java array (like `int[] days = {31,28,31,30…etc.})`?

(a) the `Array` class
(b) the `Arrays` class
(c) the `ArrayList` class
(d) The `days` arrays has its own `sort()` method; it does not need any extra methods
(e) none of the above

18. When you make a definition like:

```
Scanner scanner = new Scanner(System.in);
```

which one of the following most correctly describes what happens after this line is executed:

(a) `scanner` holds a new `Scanner` object
(b) `scanner` holds an address that points to a new instance of a `Scanner` object
(c) `scanner` holds a number that is tied to an address that points to a new instance of a `Scanner` object
(d) `scanner` holds an address that points to the `Scanner` class loaded by the class loader
(e) none of the above

19. Which *one* of the following is **not** part of the JRE (Java Runtime Environment)?

(a) The JVM
(b) Java .class files
(c) The JRE System Library
(d) Support/startup files
(e) none of the above

20. **TRUE** (a) or **FALSE** (b): Whenever any new object is instantiated, `Object`'s no-arg constructor will be loaded first, before any other constructors

21. **TRUE** (a) or **FALSE** (b): A UML diagram can describe *is a* OR *has a* relationships, but never both at the same time.

22. Which one of the following is used to indicate that a particular method is to be run as part of a JUnit test?

(a) `@Test` (b) Select `Project >> Generate JavaDoc` from the menu and follow the instructions
(c) `assertTrue()` (d) `assertFalse()` (e) none of the above

Part B: Terminology – worth 1 mark each

FILL IN THE SINGLE BEST ANSWER—ONE WORD OR SYMBOL IN THE EACH SPACE PROVIDED BELOW.

USE ONE WORD ONLY IN EACH SPACE; DO NOT USE ACRONYMS

23. Mutable variables can lead to a condition known as _____*deadlock*_____, in which two processes attempt to access the same resource simultaneously.

24. In Java the _____*diamond*_____ notation is used to indicate a *generic* type, which allows you to parameterize objects according to their class type, e.g. `java.util.arrayList<E>`

25. 'has a' relationships are of two types: __*composition*__ and ___*aggregation*____.

26. An identifier used as an argument in a method declaration is known as a *formal* parameter; an identifier passed as an argument to the same method during execution is known as a(n) _____*actual*_____ parameter. (The same terminology is used with generic types.)

27. When an inner class is not instantiated first into a named object, but instead is passed directly into a method without an identifier, it is referred to as an _____*anonymous*_____ inner class.

28. After an object has been instantiated, setting its value to _____*null*_____ will invoke garbage collection

Part C: Written Answers

Part C: Short Written Answers – ANSWER ANY 2 QUESTIONS OUT OF THE FOLLOWING 3: ONLY THE FIRST 2 QUESTIONS ANSWERED WILL BE MARKED

29. The format of the statement `Math.PI` tells us something about its declaration in the Java `Math` library. Write out the full Java code for (1) the declaration of the `Math` class and (2) the declaration of the `PI` field within that class, as you would expect to see it written in the `Math` library.

```
public final class Math {
   public static final double PI = 3.14…;
}
```

Note: no marks lost for missing 'final' in the class header since, while this is in the notes, it isn't apparent from Math.PI. However, everything else is.

Part C: Written Answers

Part C: Short Written Answers – ANSWER ANY 2 QUESTIONS OUT OF THE FOLLOWING 3: ONLY THE FIRST 2 QUESTIONS ANSWERED WILL BE MARKED

30. Describe, in general terms, why the stack is used for short-term local data, and the heap is used to store objects.

The stack uses dedicated hardware built into the CPU; this is convenient for quickly pushing and popping small, short-term variables like primitive types and reference values passed as parameters. By contrast, the heap relies on calls to the OS, which is computationally costly. Hence the heap is used to store larger objects, which are generally stored for longer periods of time.

Part C: Written Answers

Part C: Short Written Answers – ANSWER ANY 2 QUESTIONS OUT OF THE FOLLOWING 3: ONLY THE FIRST 2 QUESTIONS ANSWERED WILL BE MARKED

31. Explain why the @Override annotation is *not* strictly speaking needed to check the signature of a concrete method, if that method overrides an abstract method in the superclass.

Abstract methods in a superclass must *be overridden in the first concrete subclass. So if you make a mistake writing the method signature in the subclass, the compiler will catch it automatically—whether you use @Override or not.*

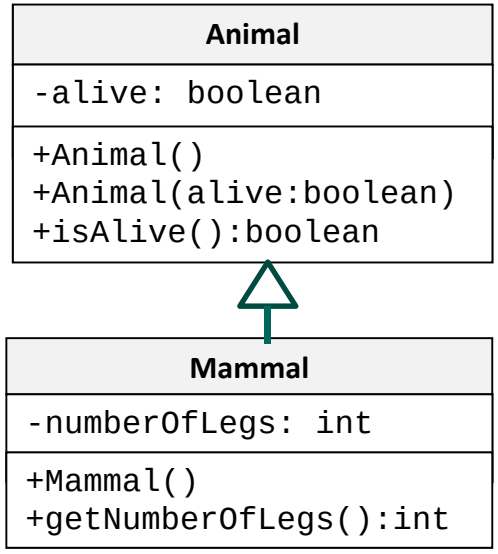
32. For the UML diagram shown below left, fill in the appropriate Java code in the boxes at right. Note that not all of the empty spaces need to be filled; ONLY ENTER WORDS IN SPACES WHERE THEY ARE INDICATED IN THE UML DIAGRAM. ***For each word entered into spaces that should be left empty, 1 mark will be deducted***. Also

(a) assume all animals are alive by default and they all have four legs.

(b) where possible, chain no-arg constructors to other constructors appropriately

(c) use only the space provided for the UML information given; do not add extra fields and constructors

(d) invent suitable parameter names where needed




```
public __________class Animal_________{
   ___private boolean alive____;
   _public_Animal(___________________){
        ____this (true)____;
    }
   _public Animal(_boolean alive_){
        ___this.alive = alive___;
   }
   _public boolean isAlive(_______){
        ______return alive ;___
   }
}
```

```
public class Mammal extends Animal{
   _______private int numberOfLegs_;
   __public Mammal(_______________){
    _________numberOfLegs = 4_;
   }
   _public int getNumberOfLegs(____){
     __________return numberOfLegs;
  }
}
```

35. The code on the next three pages contain no fewer than 20 errors, which could be manifested as either compile-time warnings or run-time errors. The latter may be due to logic errors on the part of the programmer. Your job is to find 12 of those errors. Place a number from 1 to 12 next to each of these errors, circle each of these numbers, and explain why the code is in error in the box on page 9, next to the number you just circled in the code.

Description: The following program* consists of five classes, highlighted in bold. A **Shape** class stores the `x` and `y` coordinates of a `Shape` object. It contains an abstract method `getArea()`. Additionally, it contains the `inputAllFields()` method, that prompts the user to input the `x` and `y` coodinates. **`Circle`** and **`Rectangle`** are two classes that extend `Shape`. They each override `getArea()` with an appropriate calculation. Additionally, they override `inputAllFields()` and `toString()` in the superclass and append their own information on to these methods. The **`Input`** class contains two static methods that prompt the user for input, display a string output, and then checks to see that the input is between two appropriate ranges before prompting the user for input; otherwise, the user is prompted to re-enter the values. Finally, the **`TestShapes`** class runs the `main()` routine, which prompts the user to input some number of shapes, and prompts the user for the coordinates and measurements of each shape, before finally calculating each result. Note that for simplicity, the only shapes used are `Circles` and `Rectangles`.

*From code originally supplied by Rex Woolard

```
import java.util.ArrayList;
import java.util.Scanner;
public class TestShapes {
   public static void main(String[] args) {

   ArrayList<Shape> listShapes = new ArrayList();
   int numShapes = Input.getInt("Input number of shapes:", 1, 10)

   for (int i = 0; i < numShapes; ++i) {
      Shape shape;
      if (Input.getInt("\nChoose 1:Circle  2:Rectangle", 1, 2) == 1)
         shape = new Circle();
      else
         shape = new Rectangle();
      shape.inputAllFields();
      listShapes[i] = shape;
   }

   System.out.println("All shapes found? " + (i==numShapes));
   System.out.print("Calculating Area of Shapes\n");

   for (Rectangle rect : listShapes)
      System.out.println(rect);
   }
}
```

1. ArrayList missing diamond brackets after new ArrayList; should be `new ArrayList<>();`
2. Missing ; at end of line
3. `listShapes` is an `arrayList`; it uses `.add`, not `[i]`
4. cannot use `i==numShapes`; `i` out of scope outside `for`
5. must use `Shape shape`, not `Rectangle rect`, since a `Rectangle` cannot hold a `circle` object

```
public abstract class Shape {
   private static final double MAX_X = 1920.0, MAX_Y = 1080.0;
   private double x, y;
   public Shape() { }

   public abstract double calcArea;

   public void inputAllFields() {
        x = input.getDouble("Enter x coordinate:", 0.0, MAX_X);
        y = input.getDouble("Enter y coordinate:", 0.0, MAX_Y);
   }

   @Override
   public String toString() {
      System.out.println((this instanceOf Circle)?
       "Circle“ :"Rectangle" +
        " coordinates are (" + x, + y + "), area = " + calcArea();
   }
}
```

6 `calcArea` needs () after identifier

7 Should be `Input`, not `input`

8 prints `String`, but needs to return `String`

9 `instanceof` not `instanceOf`

10 conditional `if` uses : not ;

11 needs +","+

12 missing final)

```
public class Circle extends Shape {

  private static final double MIN_RAD=1.0;
  private static final double MAX_RAD=1000;

  @Override
  public void inputAllFields(){
    super.inputAllFields();
    double radius = Input.getDouble("Enter
         radius:", MIN_RAD, MAX_RAD);
  }

  @Override                    13
  public abstract double calcArea() {
     return 2* Math.PI * radius;   14
  }

  @Override
  public String toString() {
    return(super.toString() + "radius: " +
       radius);
  }          15
}
```

13 `calcArea` declared as abstract in superclass; needs to be concrete in subclass

14 Calculates perimeter, not area

15 `radius` declared in `inputAllFields` but is only available there. The declaration for `radius` needs to be at the top of the class, so the `radius` will have global scope inside the entire class

```
public class Rectangle extends Shape {
  private static final double MIN_DIMEN = 1.0;
  private static final double MAX_DIMEN= 1000;

  private double length, width;

  @Override
  public void inputAllFields() {
16  inputAllFields();
    length = Input.getDouble("Enter length:",
         MIN_DIMEN, MAX_DIMEN);
    width = Input.getDouble("Enter width:",
         MIN_DIMEN, MAX_DIMEN);
  }

   @Override
   public double calcArea() {return x * y;}
                              17

   @Override
   public String toString() {
         return (super.toString() + " length: "
       + length + "  width: " + width);
   }
}
```

16 Needs `super`, otherwise `this` is used and the code calls itself forever

17 Supposed to return area, instead it multiplies the x and y coordinates; Should be length* width

```
public class Input {  18  19
   private Scanner input = new Scanner(System.in);
   private static double d;
   private static int i;

   public static double getDouble(String s, double min, double max){
      System.out.print(s + "("+ min + "-" + max + ")");
      do{
         d = input.nextDouble();
         input.nextLine();
      } while ((d > min) || (d < max));  20
      return d;                  21
   }

   public static int getInt(String s, int min, int max){
      System.out.print(s + "("+ min + "-" + max + ")");
         do{
         i = input.nextInt();
         input.nextLine();
       } while ((i < min) || (i > max));
       return i;
   }
}
```

18 Missing `import.util.Scanner;` (It appear in `TestShapes`, where it isn't used; that declaration needs to be here)

19 `input` needs to be `static`, otherwise you can't save the `double` value returned to `d`.

20 Missing `;`

21 Logic is backwards; you should keep looping while the input value is *less than* the minimum allowed or *greater than* the maximum allowed. Either change > < to < >, or change || to &&