CS 1331 – Final Exam Review Worksheet Solutions

NOTE: THIS IS NOT A PRACTICE EXAM: It is not meant to in any way reflect the contents or format of the Final Exam. This is a practice worksheet and is not meant to be the sole preparation for the exam. Questions on this worksheet are meant to give students a better understanding of select course concepts.

1. Consider the following class definitions:

```
public class Vehicle {
  private String name;
  private int age;

  public void travel() {
    System.out.println("Traveling vehicle");
  }
}
```

```
public class Car extends Vehicle {
  private int mpg;

  public void travel() {
    System.out.println("Car traveling");
  }

  public void fill() {
    System.out.println("Filling up gas");
  }
}
```

```
public class Plane extends Vehicle {
  private int maxHeight;

  public void fly() {
    System.out.println("Plane flying");
  }

  public void land() {
    System.out.println("Plane landing");
  }
}
```

Do the following lines of code compile? Do they run? If so, what do they print?

a.
```
Vehicle v = new Car();
v.travel();
```

Car traveling

b.
```
Vehicle v = new Plane();
v.fly()
```

Does not compile – fly( ) not defined in Vehicle

c.
```
Car c = new Plane();
c.travel();
```

Does not compile – sidecasting/incompatible types

d.
```
Plane p = (Plane) new Vehicle();
p.travel();
```

Does not run - ClassCastException

e.
```
Car c = (Car) new Vehicle();
c.fill();
```

Does not run - ClassCastException

f.
```
Vehicle v = new Car();
v.fill();
```

Does not compile – fill() not defined in Vehicle

g.
```
Vehicle v = new Plane();
v.travel();
```

Traveling vehicle

2. True/False
    a. A class can extend from multiple super classes. **False**
    b. A class can implement multiple interfaces. **True**
    c. A super class can have only one subclass. **False**
    d. An abstract class can extend from another abstract class. **True**
    e. An abstract class extending from another abstract class still has to override the parent's abstract methods. **False**
    f. An abstract class can extend from a concrete class. **True**
    g. Abstract classes can have constructors. **True**
    h. Abstract classes can be instantiated. **False**
    i. Interfaces can be instantiated. **False**
    j. An interface can be the static type for a variable. **True**
    k. Interfaces can have instance variables. **False**
    l. Abstract classes can have instance variables. **True**
    m. Interfaces can have static variables. **True**
    n. Abstract classes can have private methods. **True**
    o. Interfaces can have private abstract methods. **False**
    p. Interfaces can have private static methods. **True**

3. Consider the following code:

```
ArrayList<String> list = new ArrayList<>();
list.add("1");
list.add("2");
list.add("3");
list.remove(2);
list.remove("2");
list.set(0, "0");
list.add("4");
```

What does list look like at the end of the execution?

[0,4]

4. What are the Big O runtimes of the following code snippets?

```
for (int i = 0; i < n; i++) {
  for (int j = n; j >= 0; j = j / 2) {
    // do something
  }
}
```

O(n logn)

```
for (int i = 0; i < 100; i++) {
  for (int j = 0; j < 200000; j++) {
    // do something
  }
}
```

O(1)

```
for (int i = 0; i < n; i++) {
  for (int j = 100; j < 25; j++) {
    // do something
  }
}
```

O(n)

5. Which of the following is true about generics?
    a. Generic type parameters can be either primitives or reference types **False**
    b. Generic type checking is checked at compile time **True**
    c. You can directly instantiate a variable of a generic type **False**
    d. Generic type parameters must always be reference types **True**
    e. Generic types can be used for return values, parameters, and instance variables **True**
    f. Generics only work at the class level **False**
    g. The ! Symbol is used to denote generic wildcards **False**
    h. The expression <? extends T> means “T, or a subclass of T” **True**
    i. When code is compiled, the generic types are removed from the code **True**
    j. When code is compiled, the generic type is always replaced with Object **False**

6. Perform 3 iterations of insertion sort to sort this array in ascending order (smallest to largest)

3	2	7	6	4	1	2

Answer:

2	**3**	7	6	4	1	2
2	**3**	**7**	6	4	1	2
2	**3**	**6**	**7**	4	1	2

7. What does mystery(48) return?

```
public static int mystery(int n) {
  if (n <= 0) {
    return 10;
  }
  return mystery(n / 10) + mystery(n / 12);
}
```

40

8. Which of the following are valid ways to handle a checked exception?
   - **a. Use a try-catch block**
   - b. Switch statements
   - c. If/else blocks
   - **d. Throwing the exception**
   - e. You don't need to handle a checked exception

9. You can throw unchecked exceptions.
   - **a. True**
   - b. False

10. Explain what the code below does, and fill in the missing code:

```
try {
  Scanner scan = new Scanner(new File("test.txt"));
  PrintWriter writer = new PrintWriter("SomeFile.txt");
  int num = 0;
  while (scan.hasNextLine()) {
    num++;
    String str = scan.nextLine();
    if (str.contains("hello!")) {
      writer.println(num + ": " + str.toLowerCase());
      writer.flush();
    }
  }
  writer.close();
} // ***missing code***
  ex.printStackTrace();
```

Finds lines containing "hello" in text.txt

Writes discovered lines to someFile.txt in all lowercase with the original line number

Missing line: catch (FileNotFoundException ex) {

11. You can use lambda expressions to implement any interface
    a. True
    b. False

12. Consider the following code:

```
public void start(Stage primaryStage) {
  Button btEnlarge = new Button();
  btEnlarge.setOnAction(new EnlargeHandler());
}
class EnlargeHandler implements EventHandler<ActionEvent> {
  public void handle(ActionEvent e) {
      System.out.println("Click!");
  }
}
```

How would you rewrite the inner class EnlargeHandler as an anonymous inner class? As a lambda expression?

Anonymous Inner Class

```
bt.setOnAction(new EventHandler<ActionEvent>() {
  public void handle(ActionEvent actionEvent) {
    System.out.println("Click!");
  }
});
```

Lambda Expression

```
bt.setOnAction(e -> System.out.println("Click!"));
```

13. Given the following code, sketch the GUI

```
public void start(Stage primaryStage) {
  primaryStage.setTitle("Final Exam");
  BorderPane bp = new BorderPane();
  Label label = new Label("Good luck on your final!");
  bp.setTop(label);
  StackPane sp = new StackPane();
  Rectangle rect = new Rectangle(50, 50, Color.WHITE);
  Label label2 = new Label("yuh");
  sp.getChildren().add(rect);
  sp.getChildren().add(label2);
  bp.setCenter(sp);
  Button b = new Button("Click for good luck");
  bp.setBottom(b);
  bp.setAlignment(label, Pos.CENTER);
  bp.setAlignment(b, Pos.CENTER);
  Scene scene = new Scene(bp, 300, 300);
  primaryStage.setScene(scene);
  primaryStage.show()
}
```