

# ISE 208 Midterm Exam 2

SBCS Practice Version – SOLUTIONS

Fall 2009

NAME (please print legibly): \_\_\_\_\_

Your University ID Number: \_\_\_\_\_

- Please leave at least one seat between yourself and the person next to you.
- No books or notes can be used during the exam, except for your “cheat sheet”.
- Any **CHEATING** will result in an F as well as being written-up on academic dishonesty.

**NO BS bonus:** If you do not know the answer to a problem and leave it blank you will receive 1 point for each sub-part (e.g., a,b,c, etc.) that you leave blank. If you write anything in the space and it is wrong, you will receive a zero. Not all questions are of equal difficulty/points value, so read through the entire exam before beginning!

You will not lose any points for small coding details such as misspelling a method name, leaving out some default method arguments, or getting the order of method arguments wrong.

QUESTION	VALUE	SCORE
1	9	
2	15	
3	6	
4	10	
5	10	
6	5	
7	5	
<b>TOTAL</b>	<b>60</b>	

**1. (9 points)**

Write a method that uses recursion to count the number of times a specific character occurs in an array of characters.

```
int countCharacters(char [] arr, char match, int currPosition) // 1 pt for header
{
    int count = 0;

    if (currPos >= arr.length) // Base case -- 3 points
    {
        return 0;
    }
    if (arr[currPosition] == match) // 3 pts for this case
    {
        count = 1;
    }
    return count + countCharacters(arr, match, currPosition+1); // 2 pts for this
}
```

## 2. (15 points)

Use the following class definitions and variable declarations to answer the question that follows.

```
public class Weather
{
    // other methods omitted
    public void report ()
    {
        System.out.println("No Warnings or watches.");
    }
}
```

```
public class HighWind extends Weather
{
    // other methods omitted
    public void report ()
    {
        System.out.println("Wind Advisory.");
    }
}
```

```
public class StormWatch extends Weather
{
    // other methods omitted
}
```

```
public class TornadoWarning extends StormWatch
{
    // other methods omitted
    public void report ()
    {
        System.out.println("TORNADO WARNING!!");
    }
}
```

```
public class MyWarning
{
    public TornadoWarning bigWind;
    // other methods omitted
}
```

```
public Weather w;
public HighWind hw;
public StormWatch sWatch;
public TornadoWarning tWarn;
public MyWarning warn;
```

Which of the following assignment instructions are valid? List **ALL** of the correct answers!

- (a) `w = hw;`
- (b) `w = sWatch;`
- (c) `sWatch = warn;`
- (d) `warn = tWarn;`
- (e) `tWarn = hw;`
- (f) `warn.bigWind = tWarn;`
- (g) `tWarn = warn.bigWind;`
- (h) `w = warn.bigWind;`

ANSWER: \_\_\_\_\_a, b, f, g, h

### 3. (6 points)

Look at the following code, which is the first line of a class definition:

```
public class Tiger extends Felis
```

In what order will the class constructors execute? Why?

Felis (superclass), then Tiger (subclass) (technically, Object executes first).

**4. (10 points)**

Describe the difference between **public**, **protected**, and **private**.

Public variables/methods are accessible everywhere. Protected variables/methods are only accessible by the owning class and its descendants. Private variables/methods are only accessible to the defining class.

**5. (10 points)** You can think of this code as being “protected” because the application will not halt if it throws an exception: (5 points)

- (a) try block
- (b) catch block
- (c) finally block
- (d) protected block

ANSWER: \_\_\_\_\_A

True or false: When an exception is thrown by code inside a **try** block, the remaining statements in the **try** block are also executed. (5 points)

ANSWER: \_\_\_\_\_False

**6. (5 points)**

We wish to make the following method handle exceptions:

```
public int getAge() {  
    return this.Age();  
}
```

How would we accomplish this?

(a) 

```
throw {  
    public int getAge() {  
        return this.Age();  
    }  
    catch (Exception e) {  
    }  
}
```

(b) 

```
try {  
    public int getAge() {  
        return this.Age();  
    }  
    catch (Exception e) {  
    }  
}
```

(c) 

```
public int getAge() {  
    throw {  
        return this.Age();  
    }  
    catch (Exception e) {  
    }  
}
```

(d) 

```
public int getAge() {  
    try {  
        return this.Age();  
    }  
    catch (Exception e) {  
    } }  
}
```

ANSWER: \_\_\_\_\_D

**7. (5 points)**

Briefly explain why you might want to use a Java interface instead of normal inheritance.

Inheritance allows classes to share code and behavior, but it forces them to be related in some way. Java interfaces allow us to share behavior (in the form of identical methods) between classes that are not (and should not be) related to each other. As an example, a bicycle and a horse can both be ridden, but they do not otherwise relate to one another (one is a vehicle, while the other is an animal). In this case, we can use an interface to add a `ride()` method to both of these classes.