Steven Nguyen

Jake Nugent

CIS 368

Assignment 2

Q1:

```
package assignTwo;

/**
 * assignment2
 */
public class assignment2 {

    public static void main(String[] args) {
        MyDateClass obj = new MyDateClass();
        //set date values
        obj.setDate(5,25,00);
        // display date
        System.out.println(obj.getDate());
        // display name
        System.out.println("Steven and Jake");
    }
}
class MyDateClass{
    static int month, day, year;
    public static void setDate(int x, int y, int z){
        month=x;
        day = y;
        //converting to full year
        if (z>24)
            year = 1900+z;
        else
            year = 2000+z;
    }
    public static String getDate(){


        String giveBack =  month + "/"+ day + "/" + year;
        return giveBack;
    }
}
```

Q2 :

Wrapper classes are classes that make it possible to use primitive data types, like an int or a Boolean, as objects.

Boxing is a conversion within the Java compiler that is able to convert primitive data types to their respective wrapper class. Character cc = ‘e’; is an example of this. The char ‘e’ is autoboxed into the Character object wrapper class.

```
Interger[] intarray ={1,2,3}
```

Unboxing works the opposite of the autoboxing. It takes the object of a wrapper class and converts it to its respective primitive value. The following is an example:

```
Integer num = new Integer(258);
int temp = num;
```

The example automatically converts the Integer object wrapper class to an int with the same value and stores it in temp.

AutoBoxing and AutoUnboxing are just other names for boxing and unboxing within Java. The “auto” means that there is no other conversions needed for it to occur. However, within Java, the conversion happens automatically.

```
Int i = 1; //autoboxing

Integer j = i;//autoboxing

Int fun = j;//autounboxing
```

Q3 :

One difference between StringBuilder and StringBuffer is that a StringBuffer is thread-safe. If we want to use synchronized methods, then we should use a StringBuffer because it is thread-safe. Even with thread-safety going toward StringBuffer, StringBuilder does have one thing over StringBuffer. A StringBuilder is more efficient. Because a StringBuilder is not synchronized, it is faster. Therefore if we are using a single thread, then using a StringBuilder is preferable because being synchronized is overkill and not needed. Another difference is when both were introduced. StringBuffer was introduced in Java 1.0 and StringBuilder was introduced in Java 1.5.

Q4 :

(1)Stack memory is mainly used when in pulling off executions of threads and for static memory allocation. Heaps, on the other hand, are mainly used dynamic memory allocation especially with Java objects and JRE classes at runtime.

(2)For stack memory, all variables inside exist for as long as the method it is in is running. Stack also automatically allocates and deallocates memory when the execution is done. Compared to heaps, the access to the memory is quite fast. If we want to threads, then use stacks cause they are threadsafe. Full memory makes Java throw java.lang.StackOverFlowError.

(3)For heaps, Garbage Collector is needed to free up objects that are unused because it is not able to do automatic deallocation. Heaps need to be properly synchronized within the code because it is not safe for threads. When the space of the heap becomes full, Java ends up throwing java.lang.OutOfMemoryError.

(4) Stack memory is managed by the CPU automatically. Heap memory is allocated by programmers and can possibly cause memory leaks and memory errors.

(5)Stack memory is used for storing variables (local) and return addressing. Heap memory is used for dynamically storing objects, arrays, and strings.

Q5 :

The equals() method and the == operation act different depending if working with primitives and objects.

For primitives like ints, doubles, Booleans, and chars, they work the same. They will just compare the values of the two primitives. This works with both being the same primitive type.

For object, the equals() method and the == operation do two separate things. As far as the method goes, it will compare value of the object. This is what is most commonly used between two strings. Since strings are objects, using the method checks if they hold the same value. This can work between two objects of the same type if all of the variables of said objects are the same. The == operation, on the other hand, will just compare the memory addresses of the two objects. If the objects have the same memory address then it will return true but otherwise it will return false even if the values within them are the same.

```
classData x = new classData("X", 30);
classData y = new classData("X", 30);
if  x == y) {
    System.out.println( x and y reference the same object");
} else {
    System.out.println( x and y reference different objects");//prints points @
//diff data
}

if  x.equals(y)) {
    System.out.println( x and y have the same value");//prints
} else {
    System.out.println( x and y have different values");
}
```

Q6 :

Java supports single, multilevel, and hierarchical as the inheritance types. However, the inheritance types that are not supported are multiple and hybrid inheritance.

Q7:

Overriding is when there is a method in both the superclass and child class that have the same method signature. Overloading, on the other hand, is just like overriding but the parameters are different.

1: overloading increase the readability of code wile overriding is used to provide more specific implementation of a method that already exists in the parent class

2: overloading is preformed in a class while overriding happens in two classes that one is the child of the other.

3: Overriding, the parameter must be the same, Overloading must have different parameters.

4: Overloading provides multiple methods with the with differing functions based on the parameters. Overriding will effects the entire behavior of a method.

5: Overloading has poorer performance due to polymorphism, Overriding has better performance because of the binding of methods done at runtime.

Q8 :

One difference between interfaces and abstract classes are the types of methods that are within them. An interface can only have abstract methods. Surprisingly, abstract classes can have both abstract and non-abstract classes.

Another difference comes from final variables. An interface automatically sets the declared variables final by default. Abstract classes, on the other hand, may contain non-final variables.

The type of variables is another difference. Interface variables are only static and final. Abstract classes can have final and static variables but also can have non-final and non-static variables.

There are also differences in implementations. Interfaces can be extended to one or more interfaces. Abstract classes are able to extend to another Java class as well as implement a multitude of Java interfaces.

Finally, there are also differences with inheritance. Abstract classes are unable to support more than one inheritance. Interfaces, however, support more than one inheritance.

Q9 :

A superclass' constructor is implicitly called by the subclass if the keyword super() is not used. the superclass constructor is explicitly called by the subclass if the keyword super() or this() is used.