Name:_____

# ICS4U0 – Test - Objects

## Application (32 marks)

An iphone XS is available with a 5.8 – inch display starting at $1379 and a 6.5 – inch display starting at $1519. It comes in three colours: silver, space grey, and gold. The phone comes with 64 GB of memory. An upgrade to 256 GB is $210 and an upgrade to 512 GB is $480. AppleCare is available for $249.

1. In Java, define and create a class definition of type *iPhoneXS* based on the description above. Make sure to include the appropriate public/private keywords and variable types. Include all necessary fields (there should be five): (A – 5 marks)
   **[1 mark for each variable]**

```
public class iPhoneXS
{
    private double price;
    private double screen;
    public String colour;
    private int memory;
    private boolean appleCare;
}
```

2. Constructors

a. Create a default constructor (meaning no parameters) that will create a silver iPhone that is as cheap as possible. We'll consider this the "base" model. (3 marks)
**0.5 for each assignment**
**0.5 for constructor signature**

```
public iphoneXS()
{
    this.price = 1379;
    this.screen = 5.8;
    this.colour = "silver";
    this.memory = 64;
    this.appleCare = false;
}
```

b. Create a constructor that receives size, colour, and memory. This constructor will create an iPhone. AppleCare will not be selected for this iPhone. (3 marks)
**[assigning size, colour, memory - 1 mark]**
**Initializing the price - 2 marks**

```
public iPhoneXS(int size, String colour, int memory)
{

```

Name:_____

**ICS4U0 – Test - Objects**

```
    this.size = size;
    this.colour = colour;
    this.memory = memory;
    this.appleCare = false;
    if (size == 5.8) this.price = 1379;
    else if (size == 6.5) this.price = 1519;

    if(this.memory == 256) this.price += 210;
    else if(this.memory == 512) this.price += 480;
}
```

c. Create a constructor that receives a parameter of type iPhone. This will create an object with the same values as the iPhone parameter passed in. (3 marks)

```
public iPhoneXS (iPhoneXS phone)
{
    this.price = iPhoneXS.price;
    this.size = iPhoneXS.size;
    this.colour = iPhoneXS.colour;
    this.memory = iPhoneXS.memory;
    this.appleCare = iPhoneXS.appleCare;
}
```

3. Create accessor methods for one instance field. You may assume that accessor methods have been created for the rest. (A – 1 marks)

```
public int getPrice()
{
    return this.price;
}
```

4. Create the mutator methods for all NECESSARY FIELDS. Remember, you can upgrade OR downgrade any options (A – 10 marks)

```
void String setColor(String color){  [1 mark]
    this.colour= colour;
}

public void changeScreen(int size) [1 mark]
{
    if(this.size != size)
    {
            //increase the price
```

Name:_____

**ICS4U0 – Test - Objects**

```
            if (size ==6.5) this.price += 1519 - 1379;
          // decrease the size
          else { this.price -= 1519-1379;
}

public void changeMemory (int memory) [6 marks]
{
    if(this.memory == 64) //only possibility is that you upgrade
    {
          if(memory == 256) { this.price += 210; this.memory = memory;
          } else if (memory == 512) { this.price += 512; this.memory = memory;
     } else if (this.memory == 256)
     {
          if(memory == 64) { this.price += 210; this.memory = memory;
          } else if (memory == 512) { this.price += 270; this.memory = memory;

      } else if (this.memory == 512) //downgrade options: 64 or 256
           if(memory == 256) { this.price -= 270; this.memory = memory;
          } else if (memory == 64) { this.price -= 480; this.memory = memory;

     }


}

public void double addAppleCare()  [1 mark]
{
    this.appleCare = true;
    this.price += 249;
}

void double removeAppleCare() [1 mark]
{
    this.appleCare = false;
    this.price -= 249;
}
```

5. Explain what class field will not need a mutator method? Explain why? (A- 2 marks)

Price - you cannot directly change this value. It is dependent on changing the screen size, up or downgrading the memory, or adding / removing appleCare. They are all related.

6. Create an instance method called PrintSpecs. This method will print all the specifications for the implicit object. (2 marks)

Name:_____

**ICS4U0 – Test - Objects**

```
public String PrintSpecs ()
{
    String specs = "The iPhone has a screen size of " + this.size + " inches";
    specs += "\nwith " + this.memory + " GB."  + "It is " + this.colour + " and costs $";
    specs += this.price + ".";

    if (this.appleCare == true) specs += "It has appleCare";
    else specs += "It does not have appleCare";

   return specs;
}
```

7. Create a class method called ComparePrice that receives two parameters of type iPhone. This method will return the cheaper iPhone (or the first if it's the same). (3 marks)
   1- method signature
   2 - comparison and the return

```
public static iPhoneXS comparePrice (iPhoneXS x, iPhoneXS y)
{
    if (x.price < y.price) return y;
    return x;
}
```

**Knowledge and Understanding (16 marks)**

8. In your main program write the code to create the following objects. **Illustrate the objects that are created. If an error has occurred explain the error.**

   a. Create a base model iPhone called *x*. [2 mark]

   ```
   iPhoneXS x = new iPhoneXS();
   ```

   b. Create a silver 6.5-inch iPhone called *y* with 256 GB. [2 mark]

   ```
   iPhoneXS y = new iPhoneXS(6.5, "silver", 256);
   ```

   c. Create an iPhone called *z* that is a copy of x. [2 mark]

   ```
   iPhoneXS z = x;
   ```

Name:_____

**ICS4U0 – Test - Objects**

d. Create a variable of type iPhone called *a*. Assign *a* to *z*. [2 mark]

```
iPhoneXS a = z;
```

e. Add Applecare to *z*. [1 mark]

```
z.addAppleCare();
```

f. Change the memory of *x* to 512 GB. [1 mark]

```
x.changeMemory(512);
```

g. Change the price of *y* to $1519. [2 mark]

This cannot be done directly. This is because the price is dependent on may factors: apple care, memory, etc.

h. Change the colour of *a* to Space Grey. [1 mark]

```
a.serColour("Space Grey");
```

9. Write the code that will determine and print the cheapest iPhone. [3 marks]

```
iPhoneXS a = new iPhoneXS();
iPhoneXS b = new iPhoneXS(5.8, "silver", 512);
iPhone XS c = comparePrice(a,b);
```

**Part 2:**

**TIPS (Level)**

10. Based on the class iPhone (and its fields), create multiple menu screens that represent a Java main program that will make use of all the fields and methods. Provide various options in a graphical application and describe how your program will work and what you want it to do. Ensure that all constructors, instance methods and class methods are used. You do not need to write a program but you should explain each screen and where the various options take you. In addition, think of one additional feature to add to your application. It should be represented by a field (variable) and a new method in your program that

Name:_____

**ICS4U0 – Test - Objects**

gives the user a new option that enhances their experience overall when deciding to buy a new phone. It can be a service, new phone feature, or other option but it should give the user more options and/or flexibility. You can go to RT → NEW. Then you can click on the SHAPE option and provide elements that show what the menu screens will look like. There is also a text tool so you write the options directly into the various shapes. You can also import pictures but the screens should be easy to understand and all options must be accounted for.

Things to look for:

- default constructor (cheapest phone)
- constructor (size, colour, memory)
- accessor method for all features (colour, memory, screen, applecare, price)
- mutator methods for all features (colour, memory, screen, applecare, NOT price)
- Copy the specs of an existing iPhone
- PrintSpecs
- Compares two iPhones and returns the cheaper one

Additional Feature