I. True/False Questions. (20 points) Mark T or F as appropriate. Note: Answers to some questions may depend on the version of Java that is being used – for these questions, the JDK that is being referred to is **JDK 1.8**

__F__ 1. Static variables cannot be accessed by instance methods.

__T__ 2. When class B is a subclass of class A and it makes sense for B to make use of the equals method defined in A, the best strategy for overriding equals in A is the *instanceof strategy*.

__F__ 3. Private methods in a class `ClassA` are accessible to those classes, and *only* those classes, that live in the same package as `ClassA`.

__T__ 4. Suppose A is a class with two constructors explicitly defined, one of which is the no-argument constructor. Suppose B is a subclass of A. If B does not need an explicit constructor for its own initialization, it is not necessary to explicitly define a constructor in B since the compiler will know that the intended superclass constructor will be A's no-argument constructor.

__F__ 5. ((~5) & 6) ^ 7 has the same value as (~5) & (6 ^ 7)

__T__ 6. A member inner class has access to all fields and methods in its enclosing class.

__F__ 7. A subclass has access to all public methods and variables of its superclass, but must explicitly create an instance of the superclass in order to gain this access.

__T__ 8.
```
public class MyClass2 {
    Date d1 = new Date(5L);
    Date d2 = new Date(33L);
    void modify(Date x, Date y) {
       x.setTime(y.getTime());
    }
    public static void main(String[] args) {
       new MyClass2();
    }
    MyClass2() {
       modify(d1,d2);
       System.out.println("d1: " + d1.getTime());
       System.out.println("d2: " + d2.getTime());
    }
}
```
The output of this code when the `main` method is run is:

```
d1: 33
d2: 33
```

_T_ 9.

```
public class MyClass {
    Date d1 = new Date(5L);
    Date d2 = new Date(33L);
    void modify() {
        d1 = d2;
    }
    public static void main(String[] args) {
        new MyClass();
    }
    MyClass() {
        modify();
        System.out.println("d1: " + d1.getTime());
        System.out.println("d2: " + d2.getTime());
    }
}
```

The output of this code when the `main` method is run is:

d1: 33
d2: 33

_F_ 10. Every member inner class of a class A is also a subclass of class A.

II. **Multiple Choice.** (3 points each) Select the best answer in each case (only one answer allowed for each problem).

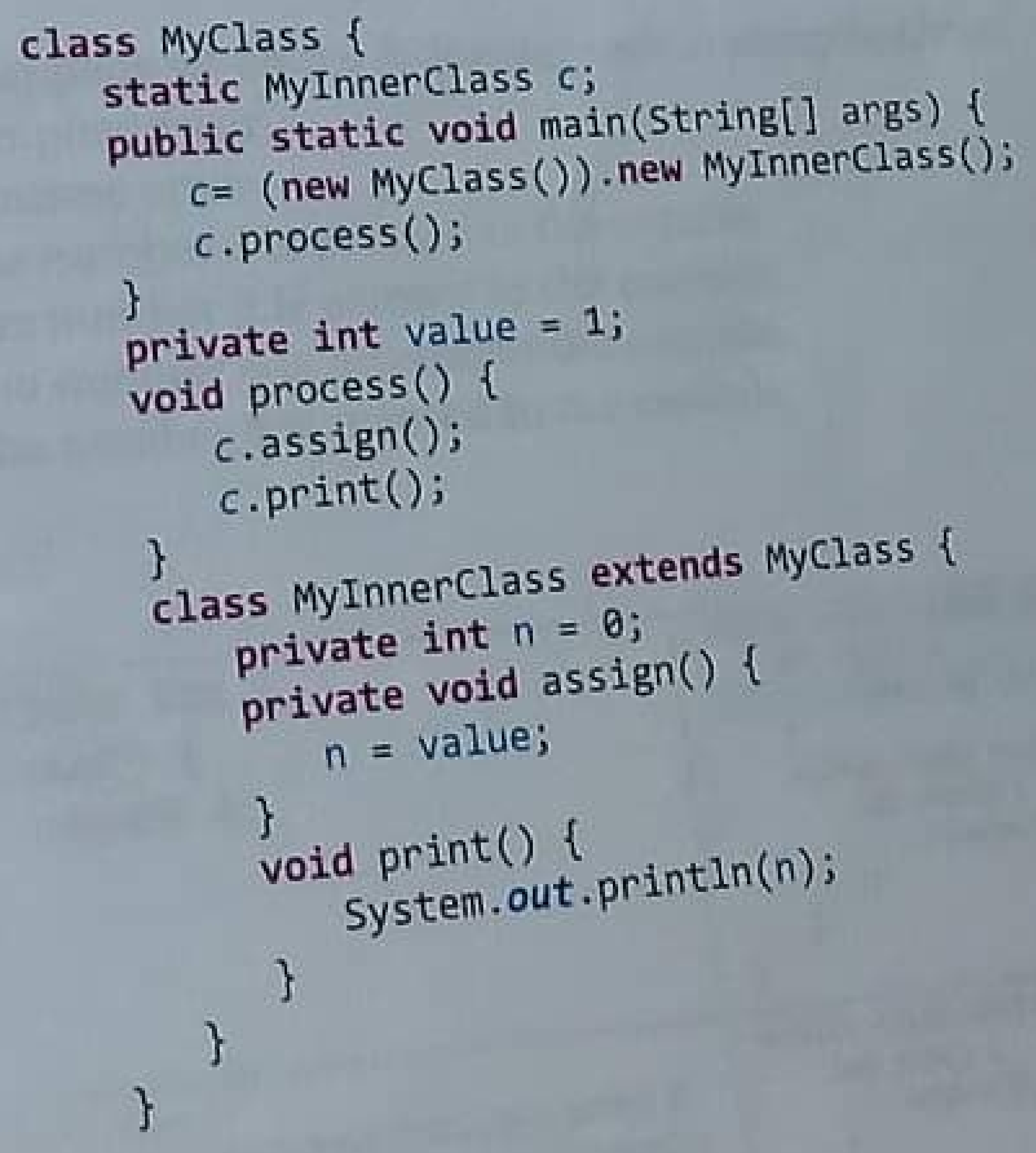
_b_ 1. What happens when you compile/run the following code:

```
class MyClass {
    public static void main(String[] args) {
            new MyClass();

    }
    MyClass() {
        AnotherClass a = new AnotherClass(this);
    }
    private void myMethod() {
            System.out.println("hello");
    }
}
class AnotherClass {
    AnotherClass(MyClass m) {
        m.myMethod();
    }
}
```

a. Outputs "hello" to the console
b. Compiler error
c. Runtime exception

_c_ 2. What happens when you compile/run the following code:



```
class MyClass {
    static MyInnerClass c;
    public static void main(String[] args) {
        c= (new MyClass()).new MyInnerClass();
        c.process();
    }
    private int value = 1;
    void process() {
        c.assign();
        c.print();
    }
    class MyInnerClass extends MyClass {
        private int n = 0;
        private void assign() {
            n = value;
        }
        void print() {
            System.out.println(n);
        }
    }
}
```

a. Compiler error
b. Runtime error
c. Outputs 1 to the console

__a__ 3. A developer wishes to create three classes A, B, C, so that C inherits from B and B inherits from A. Classes A and B need to have different `equals` methods, but the developer wishes to allow class C to inherit the `equals` method from B without overriding it. His implementation of these requirements is shown in the code below. Which of the following statements is correct (with reference to the code shown below)?

a. For any two instances c1, c2 of C, the statement c1.equals(c2) will return true.

b. Because C inherits the `equals` method from B, there will be asymmetric `equals`: It is possible for there to be an instance of C that is `equal` to an instance of B, but this instance of B is not `equal` to the instance of C.

c. Because C inherits the `equals` method from B and the `getClass` strategy is used in B, it is possible for two instances of C to be considered *not* `equal`, even if the value of `w` in each instance is the same.

```java
public class A {
    private int x = 1;
    @Override
    public boolean equals(Object ob) {
        if(ob == null) return false;
        if(getClass() != ob.getClass())
            return false;
        A a = (A) ob;
        return a.x == x;
    }
}
```

```java
public class B extends A {
    private int x = 2;
    private String y = "B";
    @Override
    public boolean equals(Object ob) {
        if(ob == null) return false;
        if(getClass() != ob.getClass())
            return false;
        B b = (B) ob;
        return b.x == x && b.y.equals(y);
    }
}
```

```java
public class C extends B {
    int w = 3;
}
```

_f_4. What happens when the following code is compiled/run?

a. Compiler error

b. Runtime error

c. The number 2 is printed to the console.

d. The number 3 is printed to the console.

e. The number 4 is printed to the console.

f. The number 5 is printed to the console

```java
public class Top {
    int num() {
        return 2;
    }
}
```

```java
public class Middle extends Top {
    int num() {
        return 3;
    }
    static class MiddleInner extends Middle {
        int num() {
            return 4;
        }
    }
}
```

```java
public class Main {
    public static void main(String[] args) {
        Middle.MiddleInner mm = new Bottom();
        Top t = mm;
        System.out.println(t.num());
    }
}
```

```java
public class Bottom extends Middle.MiddleInner {
    int num() {
        return 5;
    }
}
```

III. Short Answer (7 points)

1. What is the output when the following code is compiled/run? (No compiler errors will occur.)

```
public class MyString implements Comparable<MyString> {
    private String aString;
    MyString(String s) {
        aString = s;
    }
    @Override
    public String toString() {
        return aString;
    }
    @Override
    public int compareTo(MyString s) {
        if(aString.length() < s.aString.length()) {
            return -1;
        } else if (aString.length() > s.aString.length()) {
            return 1;
        } else {
            return 0;
        }
    }
    public static void main(String[] args) {
        MyString[] myStrings = {
                new MyString("Bob"),
                new MyString("Andrew"),
                new MyString("Charles")};

        Arrays.sort(myStrings);
        System.out.println(Arrays.toString(myStrings));
    }
}
```

```
Your Answer:

  [Bob, Andrew, Charles]
```

Grading: 7 if they get these names in this order, 0 if not

Programming (15 points)

1. Below, the class Manager inherits from Employee, and equals has been overridden in Employee using the instanceof strategy. Your development team decides Manager needs to have its own equals method that takes into account the bonus field. Your team decides to proceed by using composition instead of inheritance. Rewrite the code shown below so that composition instead of inheritance is used and each class has its own equals method (and the bonus field is used in determining equality between two Manager objects). No need to write Employee class since it will not be changed.

```
class Employee {
   private String name;
   private int salary;
   private LocalDate hireDay;

   Employee(String aName, int aSalary, int aYear,
         int aMonth, int aDay) {
      name = aName;
      salary = aSalary;
      hireDay = LocalDate.of(aYear, aMonth, aDay);
   }

   public String getName() {
      return name;
   }

   public int getSalary() {
      return salary;
   }

   public LocalDate getHireDay() {
      return hireDay;
   }

   @Override
   public final boolean equals(Object ob) {
      if (ob == null) return false;
      if (ob instanceof Employee) return false;
      Employee e = (Employee) ob;
      return (e.name.equals(name) && e.salary==salary
         && hireDay.equals(hireDay));
   }
}
class Manager extends Employee {
   public Manager(String name, int salary,
         int year, int month, int day) {
      super(name, salary, year, month, day);
      bonus = 0;
   }

   @Override
   public int getSalary() {
      int baseSalary = super.getSalary();
      return baseSalary + bonus;
   }

   public void setBonus(int b) {
      bonus = b;
   }

   private int bonus;
}
```

Solution:

No change to Employee class. New Manager class below:

```
class Manager {
   private Employee e;
   public Manager(String name, int salary,
         int year, int month, int day) {
     e = new Employee(name, salary, year, month, day);
     bonus = 0;
   }

   public String getName() {
     return e.getName();
   }

   public LocalDate getHireDay() {
     return e.getHireDay();
   }

   public int getSalary() {
     int baseSalary = e.getSalary();
     return baseSalary + bonus;
   }
```

```
    public void setBonus(int b) {
        bonus = b;
    }

    @Override
    public boolean equals(Object ob) {
        if(ob == null) return false;
        if(getClass() != ob.getClass()) return false;
        Manager m = (Manager)ob;
        return m.e.equals(e) && m.bonus == bonus;
    }

    private int bonus;
}
```

Grading

3 points for each of the following:

- No "extends Employee" in Manager
- In Manager constructor, the following line should occur as the first line:

  ```
  e = new Employee(name, salary, year, month, day);
  ```
- The getName method in Manager must be implemented using the Employee instance
- The getHireDay method in Manager must use the Employee instance
- There must be separate equals method in Employee and Manager (in Manager, either instanceof or same classes strategy is fine in this implementation)

2. In a university Art Department, there are two specializations or sub-departments: Fine Art and Web Art. Below, classes representing these departments are shown. The `WebArt` class records the web art pieces that have been created in that sub-department during a particular semester, stored in the variables `animations` and `adverisements`, and the `FineArt` class records the fine art pieces that have been created in that sub-department during a particular semester, stored in the variables `paintings` and `sculptures`.

   For this problem, you must complete the code for the class `ArtCreationData`. This class has two unimplemented methods that you must implement:

   ```
   Object[] assembleCommonArt(FineArt[] fineArtWorks, WebArt[] webArtWorks)
   int computeNumberArtworks(Object[] artWorks)
   ```

The method assembleCommonArt places all the elements of the two input arrays into a larger array having a common type and returns it. The common type shown here is Object, but this must be replaced by a more suitable common type, which will support polymorphism; a more suitable common type has been provided for you: the interface ArtCount.

The method computeNumberArtworks polymorphically computes the total number of artworks contained in the input array. The input array type that is shown by default is Object[], but this must be replaced by a suitable type that will support polymorphic computation of the total number of artworks.

A Main class is also shown in order to illustrate the flow of the application. The main method of the Main class provides a typical scenario: two WebArt instances are created, representing web art work for two semeseters, and two FineArt instances are created, representing fine art created over two semesters.

What you must do: Write your implementations of the methods in the space provided (on the page following the code shown below).

```
public class FineArt  {
    private String[] paintings;
    private String[] sculptures;
    public FineArt(String[] paintings,
            String[] sculptures) {
        this.paintings = paintings;
        this.sculptures = sculptures;
    }
    public String[] getPaintings() {
        return paintings;
    }
    public String[] getSculptures() {
        return sculptures;
    }
}
```

```
public class WebArt {
    private String[] animations;
    private String[] advertisements;
    public WebArt(String[] animations,
            String[] advertisements) {
        this.animations = animations;
        this.advertisements = advertisements;

    }
    public String[] getAnimations() {
        return animations;

    }
    public String[] getAdvertisements() {
        return advertisements;

    }
}
```

```
public interface ArtCount {
    public int numArtPieces();
}
```

```
public class Main {
    static FineArt[] fineArtsArr;
    static WebArt[] webArtsArr;

    public static void main(String[] args) {
        populateArrays();
        ArtCreationData acd = new ArtCreationData();

        //change Object[] type to something more suitable
        Object[] artObjects
          = acd.assembleCommonArt(fineArtsArr, webArtsArr);
        System.out.println("Number of pieces of art : "
          + acd.computeNumberArtWorks(artObjects));
    }

    ///////////////////test data
    private static void populateArrays() {
        fineArtsArr = new FineArt[] {
            new FineArt(paintings1, sculptures1),
            new FineArt(paintings2, sculptures2)};
        webArtsArr = new WebArt[] {
            new WebArt(animations1, advertisements),
            new WebArt(animations2, new String[0])};

    }
    static String[] paintings1
      = new String[]{"Horizon", "Sunset", "Speeding"};
    static String[] paintings2
      = new String[]{"Horizon", "Sunset", "Speeding"};
    static String[] sculptures1
      = new String[]{"Thinker","TallMan","Invincible"};
    static String[] sculptures2
      = new String[]{"Wonder Boy"};
    static String[] animations1
      = new String[]{"Sponge Bob", "Batman", "Goofy"};
    static String[] animations2
      = new String[]{"Waking Up", "Dusk", "Car Race"};
    static String[] advertisements
      = new String[]{"Cereal", "Skim Milk", "Dog Bones"};

}
```

Solution:

```
//Only modifications have been shown
public class FineArt implements ArtCount {
    public int numArtPieces() {
        return paintings.length + sculptures.length;
    }
}


public class WebArt implements ArtCount {
    public int numArtPieces() {
        return animations.length + advertisements.length;
    }
}
```

//Implement the methods shown below. Be sure to change Object types to ArtCount

```
public class ArtCreationData {
   //Creates an array of all art types;
   //return value is used in computeNumberArtWorks
   public Object[] assembleCommonArt(FineArt[] fineArtWorks,
         ArtCount[] newArray
             = new ArtCount[fineArtWorks.length + webArtWorks.length];

         //student may do these copy operations in a loop - not
         //necessary to use arraycopy
         System.arraycopy(
             fineArtWorks, 0, newArray, 0, fineArtWorks.length);
         System.arraycopy(
             webArtWorks, 0, newArray,
                 fineArtWorks.length, webArtWorks.length);
         return newArray;
   }


   //Polymorphically totals number of WebArt and FineArt pieces
   //of art and returns this number. Modify the type Object[]
   //appropriately
   public int computeNumberArtWorks(ArtCount[] artWorks) {
      int total = 0;
      for(ArtCount ac : artWorks) {
         total += ac.numArtPieces();
```

```
        }
        return total;
    }

}
```

Grading

5 points for implementing ArtCount in FineArt and WebArt
5 points for assembleCommonArt in ArtCreationData (must change Object[] to ArtCount[])
5 points for computeNumberArtWorks – must use polymorphism; Object[] must be changed to ArtCount[]

IV. **SCI Question** (3 points) In a short essay, discuss two main points from the lessons of the course so far in light of SCI. Do this for each point by first stating the point from the discipline (computer science) and then describing how this point exhibits properties or dynamics that are parallel to properties/dynamics of consciousness or creative intelligence, as discussed in SCI.