CECS 277 Midterm 1 Review

1.Implement a subclass
```
    public class B extends A {

    }
```
2. Abstract classes
    Cannot instantiate an abstract class
    the top base class in an inheritance

3. Subclass constructor

```
    public class Circle extends Shape
    {
      private double radius;
      public Circle(int r) {
        super();//call Shape()
        radius = r;
    }

      public Circle(int x, int y, int r)
     {
        super(x,y); // calls Shape(x,y)
        radius = r;
      }
    }
```

4. final
    If a method is final, then the method cannot be overridden
      in the subclass.

5. Overriding methods from a base class
    Let Circle extend Shape
```
    public class Circle extends Shape {
      .
      .
      .
      //every class has a toString() method.
      public String toString() {
        return super.toString();//super.toString() calls the method to
                                //toString in the base class
      }
    }
```

6. Implementing an abstract method in a subclass
      No key word abstract

```
7. private, protected, public
    public class A {
      protected int a;
    }

    public class B extends A {
      public void add() {
        //subclasses have access to protected
        a = a + 5;
      }
    }

    public static void main() {
    B b = new B();
    System.out.println(b.a); // THIS RESULTS IN COMPILER ERROR
    //SINCE 'a' IS PROTECTED. 'a' can only be seen by
    //subclass implementations/same packages
    }

8. Polymorphism
    Shape[] s = new Shape[10];
    s[0] = new Circle(__);
    Rectangle r = new Rectangle(__);
    s[1] = r;
    .
    .
    .
    s[9] = new Cylinder(__);
    //Assume double computeArea() is an abstract method.
    double totalArea = 0.0;
    for (int i = 0; i <= s.length; i++) {
      totalArea += s[i].computeArea();
    }

//using instanceOf to find the total area of all rectangles in array s.

    double totalAreaofRectangles = 0.0;
    for (int i = 0; i <= s.lengh; i++) {
      if (s[i] instanceof Rectangle) {
        //do something
        totalAreaofRectangles += s[i].computeArea();
      }
    }
```

```
//using instanceOf to find the total volume of all rectangles in array s.
//Assume the method computeVolume is defined only in the class Cylinder

      double totalVolume = 0.0;
      for (int i = 0; i <= s.lengh; i++) {
        if (s[i] instanceof Cylinder)
      {
          //do something
          totalVolume += ((Cylinder) s[i]).computeVolume();
        }
      }
```

9. Composition

```
      class Book {

      }

      class BookOrder {
        private Book b; // <- composition
      }
```

Note: Inheritance is an "is-a" relationship
      Composition is a "has-a" relationship

10. Copy constructor vs clone

Copy constructors

```
        public class A {
          private int a1;
          private int a2;
          .
          .
          .
          public A(A a ) {
              a1 = a.a1;
              a2 = a.a2;
          }
        }
```

Clone
  Shallow copy vs deep copy
    Shallow Copy
      No composition

```
          class A implements Cloneable {
            .
            .
            public Object clone() {
                 try {
                   return super.clone();
                 } catch (CloneNotSupportedException e) {
```

```
                return null;
              }
            }
          }
        Deep Copy
          class A implements Cloneable {

          }
          class B extends A implements Cloneable{
            private A a;
            public setA(A aa)
            { a = aa;}

            public Object clone() {
              try {
                B b = (B)super.clone();
                A a = (A)a.clone();)
                b.setA(a);;
                return b;
              } catch (CloneNotSupportedException e) {
                return null;
              }
            }
          }
```

11. UML Diagram
    -private
    +public
    #protected
12. Javadoc

```
    /**
      @param
      @return
    */
```

13. Interface
    Constant Variable
    abstract method declaration
14. Sorting

```
      public class Employee implements Comparable<Employee> {
        private int id;
        private String name;

        public int compareTo(Employee o) {
          //compare id
          return this.id - o.id;
        }
      }
   //Create another class to sort name
      public class SortByName implements Comparator {
        public int compare(Object o1, Object o2) {
            //compare name
          }
        }



        public static void main() {
          Employee[] e = new Employee[5];
          e[0] = new Employee(123,....);
          .
          .
          .
          e[4] = new Employee(423,....);

          //sorting by id since compareTo sorts by name.
          Arrays.sort(e);

          //sorting by name
          Arrays.sort(e, new SortByName());
        }
    15. Method .equals() to compare content (value) of objects
      == compare reference
      Circle c1 = new Circle(__);
      Circle c2 = c1;
      //c2 is just a pointer to where c1 is pointing.
      //c1==c2 is true
      //To actually compare content, you must override the equals() method
      public Circle {
        .
        .
        .
        public boolean equals(Object o) {
          if (o instanceOf Circle) {
            return radius == ((Circle)o).radius;
          }
       //check if radius of c1 equals c2
      if (c1.equals(c2)) {

      }

16. Exception

   Checked exception
   Unchecked exception
   try, catch and finally
   throws

17. File
    Read (Scanner) and write (PrinterWriter)
```