Recitation 3

Inner Classes, Interfaces

---

1. Inner Classes
    1. Write a class named Outer that contains an inner class named Inner. Add a method to Outer that returns an object of type Inner. Outer has a private String field (initialized by the constructor), and Inner has a toString() that displays this field. In main(), create and initialize a reference to an Inner and display it.
    2. Write a class named Outer2 that contains an inner class named Inner, and the Outer2 class itself has a method that returns an instance of the inner class. In a separate class named InnerApp, make an instance of the inner class without creating an object of the outer class.

---

2. Short questions
    1. Will the following code compile?

```
public class D { }

public class C implements Comparable<D> {
  public int compareTo(D o) { return 0; }
}
```

    2. Will the following code compile?

```
public class D { }

public class C implements Comparable<C>, Comparable<D> {
  public int compareTo(C o) { return 0; }
  public int compareTo(D o) { return 0; }
}
```

    3. Will the following code compile?

```
public class A implements Comparable<A> {
  public int compareTo(A o) { return 0; }
}

public class B extends A implements Comparable<B> {
  public int compareTo(B b) { return 0; }
}
```

    4. Will the following code compile?

```
public interface I { void stuff(); }
public interface J { void stuff(); }
public class F implements I,J { }
```

    5. Will the following code compile?

```
public interface I { void stuff(); }
public interface J { void stuff(); }
public class F implements I,J { public void stuff() { } }
```

    6. Will the following code compile?

```
public interface I { int stuff(); }
public interface J { void stuff(); }
public class F implements I,J { public int stuff() { return 3;} }
```

7. Will the following code compile?

```
class X implements Comparable<X> {
  public int compareTo(X o) {
    return 0;
  }
}
public class Searcher {

  public static <T extends Comparable<T>>
  boolean binarySearch(T[] list, T item) {
    return false;
  }

  public static void main(String[] args) {
    X[] xs = new X[2];
    xs[0] = new X();
    xs[1] = new X();
    binarySearch(xs,new X());
  }
}
```

8. Will the following code compile?

```
class X implements Comparable<X> {
  public int compareTo(X o) {
    return 0;
  }
}
class Y extends X { }
public class Searcher {

  public static <T extends Comparable<T>>
  boolean binarySearch(T[] list, T item) {
    return false;
  }

  public static void main(String[] args) {
    Y[] ys = new Y[2];
    ys[0] = new Y();
    ys[1] = new Y();
    binarySearch(ys,new Y());
  }
}
```

9. Will the following code compile?

```
class X implements Comparable<X> {
  public int compareTo(X o) {
    return 0;
  }
}
```

```
class Z implements Comparable<X> { }
public class Searcher {

  public static <T extends Comparable<T>>
  boolean binarySearch(T[] list, T item) {
    return false;
  }

  public static void main(String[] args) {
    Z[] zs = new Z[2];
    zs[0] = new Z();
    zs[1] = new Z();
    binarySearch(zs,new Z());
  }
}
```

10. Will the following code compile?

```
class X implements Comparable<X> {
  public int compareTo(X o) {
    return 0;
  }
}
class Y extends X { }
class Z extends Y { }

public class Searcher {

  public static <T extends Comparable<T>>
  boolean binarySearch(T[] list, T item) {
    return false;
  }

  public static void main(String[] args) {
    Z[] zs = new Z[2];
    zs[0] = new Z();
    zs[1] = new Z();
    binarySearch(zs,new Z());
  }
}
```

---

3. Suppose you built a Java library of sorting algorithms: insertion sort, quicksort, and heapsort. You want to sell this library. How would you package your library *using interfaces*, so users could use any of these algorithms in their applications, and switch from using one to another (*interface polymorphism*), with the least amount of code rewrite?

---

4. Aside from the java.lang.Comparable<T> interface used for comparing objects of a class, the java.util package has an interface, Comparator<T> that may also be used to compare objects. What is the difference between these two interfaces, and how may this difference be usefully employed in applications?