ECE 651
Lecture 2: Java
Notes Outline

- Point example:

```
public class Point {
 private final int x;
 private final int y;

 public Point() {
   this(0, 0);
 }

 public Point(int x, int y) {
   this.x = x;
   this.y = y;
 }

 public int getX() {
   return x;
 }

 public int getY() {
   return y;
 }

 public double distanceTo(Point otherPoint) {
   int dx = otherPoint.getX() - x;
   int dy = otherPoint.getY() - y;
   return Math.sqrt(dx * dx + dy * dy);
 }
}
```

- toString

- Objects vs primitivies

- hashcode and equals
  - Pitfalls to avoid:
- Java IO
  - Pitfalls to avoid:
- Object streams
  - Pitfalls to avoid:
- Dynamic dispatch
  - Differences between C++ and Java
  - How do differences relate to design principles?

- Memory Allocation
  - Differences between C++ and Java
  - How do differences relate to design principles?
  - Implications of no-RAII on other types of allocations:
- Exception handling in Java
  - Similarities to C++
  - finally
  - Kinds of Throwables:
  - Exception specifications:

- Try-finally (old way):

```
public void readAndPrintFile(String name) throws IOException {
  BufferedReader br = new BufferedReader(new FileReader(name));
  try {
    String s;
    while ((s = br.readLine()) != null) {
       System.out.println(s);
    }
  }
  finally {
    br.close();
  }
}
```

- Try-with-resource (better way):

```
public void readAndPrintFile(String name) throws IOException {
  try ( BufferedReader br = new BufferedReader(new FileReader(name)) ) {
    String s;
    while ((s = br.readLine()) != null) {
       System.out.println(s);
    }
  }
  //implicit finally {if (br != null) {br.close();}}
}
```

- No Multiple Inheritance
  - Interfaces
  - Relationship to design principles

- Parametric Polymorphism:
  - What is parametric polymorphism:
  - How did we see it in C++?
  - Differences in Java
  - Bounded Polymorphism
  - Relationship to design principles
- Operator Overloading

- Unit Testing (JUnit)

```
class SomethingTest {
   @Test
   public void test_someFun() {
    Something s = new Something(42);
    assertEquals(12, s.someFun(3));
    assertEquals(19, s.someFun(7));
   }
   @Test
   public void test_anotherFun() {
     Something s = new Something(99);
     assertEquals(\hello", s.anotherFun());
   }
}
```