ECE 651
Lecture 7: Design Patterns 2
Notes Outline

- Think, Pair, Share: what issues are there with this code?

```
class AnimalEnclosure {
  String animalType;
  public AnimalEnclosure(String ty) { animalType = ty; }

  public void refreshSupplies() {
    if (animalType.equals("lion")) {
       Meat m = new Meat();
       feeder.fill(m);
       Ball b = new Ball();
       toybox.empty();
       toybox.add(b);
    }
    else if (animalType.equals("dolphin")) {
       Fish f = new Fish();
       feeder.fill(f);
       Ring r = new Ring();
       toybox.empty();
       toybox.add(r);
    }
    else if (animalType.equals("elephant")) {
     //...
    }
  }
}
```

- Abstract Factory Pattern:

```
public class AnimalEnclosure {
  private EnclosureSupplyFactory supplyFactory;
  public AnimalEnclosure(EnclosureSupplyFactory supplyFactory) {
    this.supplyFactory = supplyFactory;
  }
  public void refreshSupplies() {
    if (animalType.equals("lion")) {
       Food f  = supplyFactory.createFood();
       feeder.fill(f);
       Toy t = supplyFactory.createToy();
       toybox.empty();
       toybox.add(t);
    }
 }
}
```

- Bank Logger problems:

- Singleton pattern:

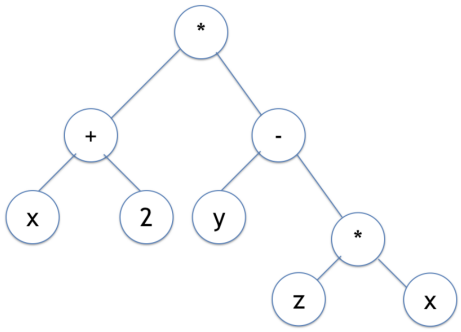
```
public class BetterLogger{
  private static BetterLogger logger = null;
  private final String logFile = "demo_better_log.txt";
  private PrintWriter writer;

  private BetterLogger() {
    try {
      FileWriter fw = new FileWriter(logfile);
      writer = new PrintWriter(fw, true);
    } catch(IOException e) { /* error handling elided */ }
  }
  public static synchronized BetterLogger getInstance() {
    if (logger == null) {
      logger = new BetterLogger();
    }
    return logger;
  }
  public void logWithdraw(String account, double amount) {
    writer.println("WITHDRAW (" + account+ "): " + "$" + amount);
  }
}
```

  - Advantages:

  - Disadvantages:

  - Appropriate uses:

- Double Checked Locking:
  - Idea:
  - Dangers:
- Expression Trees (seen in 551):




```
public abstract class ExpressionNode {
   public abstract double evaluates(HashMap<String, double> vars);
}

public class PlusNode extends ExpressionNode {
  ExpressionNode lhs;
  ExpressionNode rhs;
  public PlusNode(ExpressionNode left, ExpressionNode right) {
   lhs = left; rhs = right;
  }
  @Override
  public double evaluate(HashMap<String, double> vars) {
    return lhs.evaluate(vars) + rhs.evaluate(vars);
  }
}
```

- Visitor Pattern:

```
public abstract class ExpressionNode {
   public abstract <T> T accept(ExpressionVisitor<T> v);
}

public class PlusNode extends ExpressionNode {
  ExpressionNode lhs;
  ExpressionNode rhs;
  public PlusNode(ExpressionNode left, ExpressionNode right) {
   lhs = left; rhs = right;
  }
  @Override
  public <T> T accept(ExpressionVisitor<T> v) {
    return v.visit(this);
  }
}
```

- An evaluation visitor:

```
public class EvalVisitor implements ExpressionVisitor<Double> {
 private HashMap<String, Double> varValues;
 public EvalVisitor(HashMap<String, Double> varMap) {
   varValues = varMap;
 }
 @Override
 public Double visit(PlusNode n) {
   double d1 = n.getLeft().accept(this);
   double d2 = n.getRight().accept(this);
   return d1 + d2;
 }
 //...
}
```

- Advantages and disadvantages