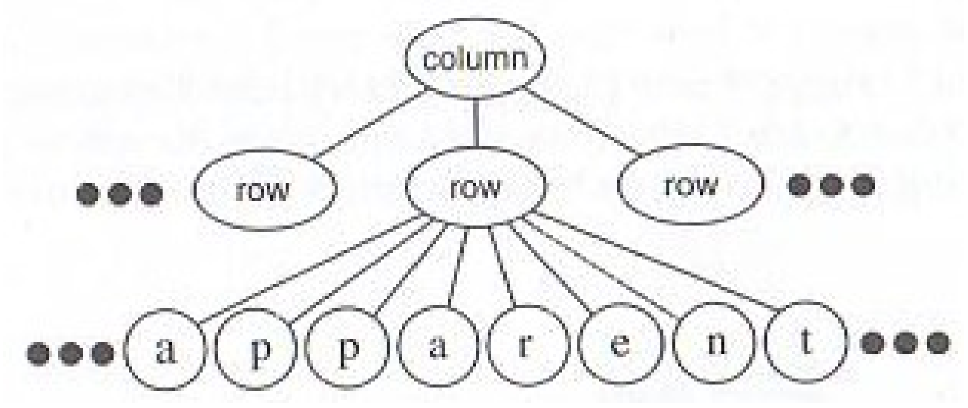
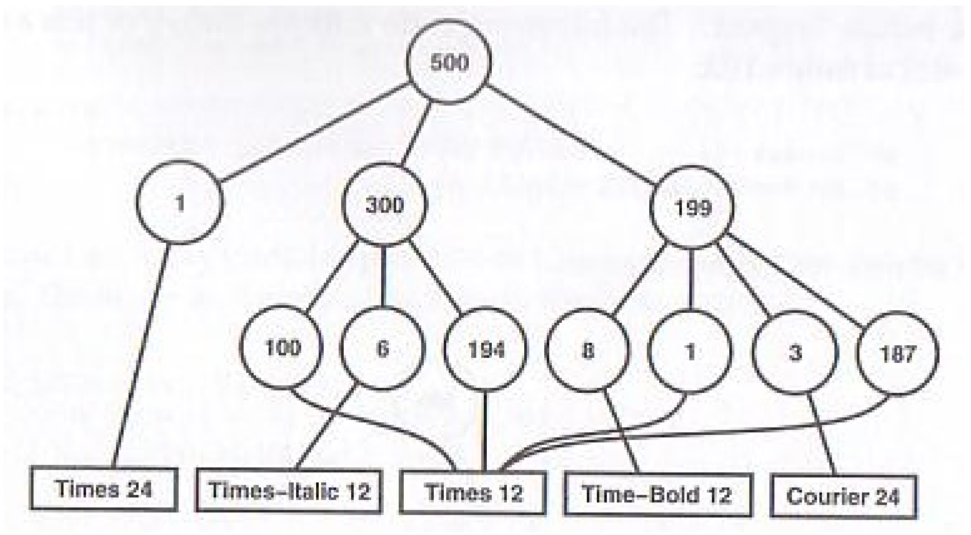
Plan for Today

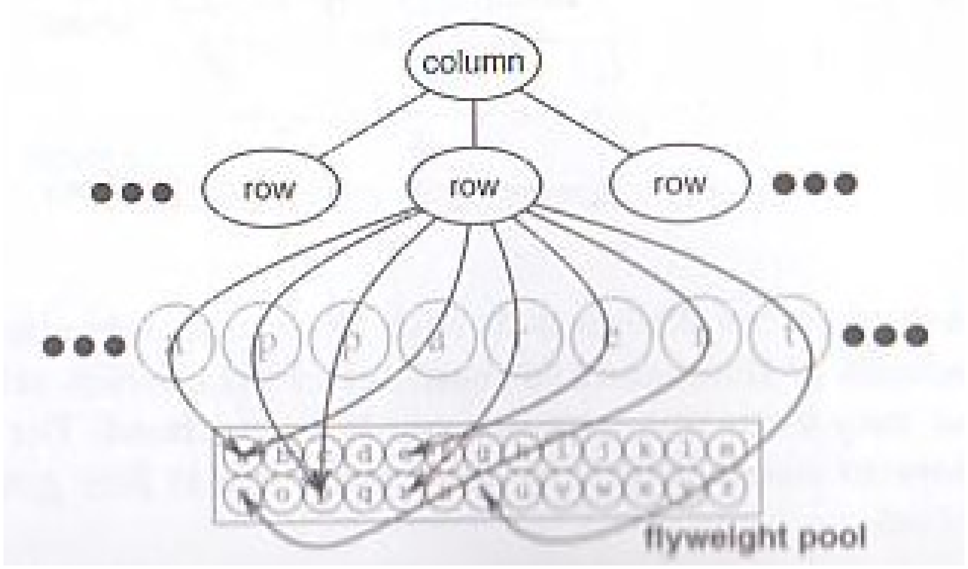
- Pre-Task 6 postmortem
- Command pattern

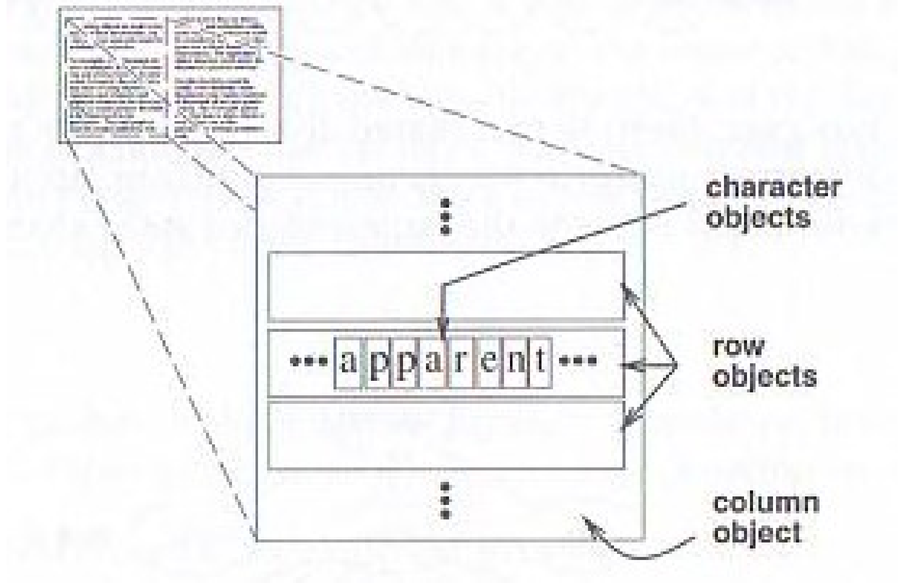
Lecture 50 – 2 December

Task 7 Questions?

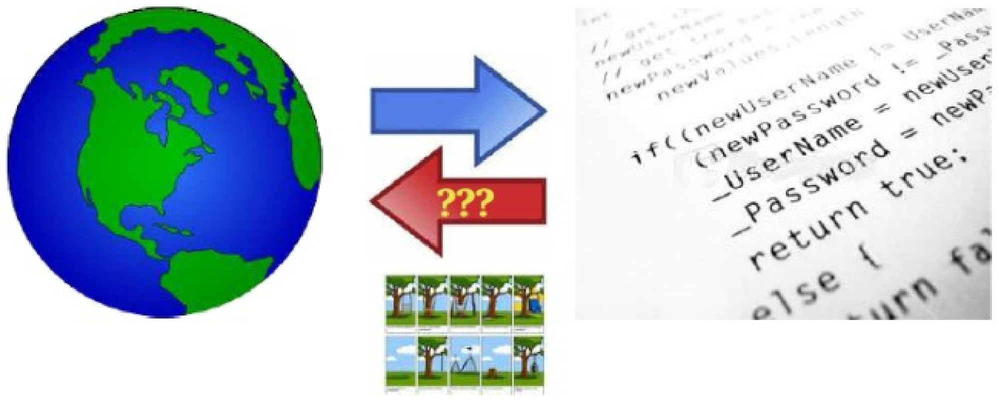








Pre-Task 6 Postmortem



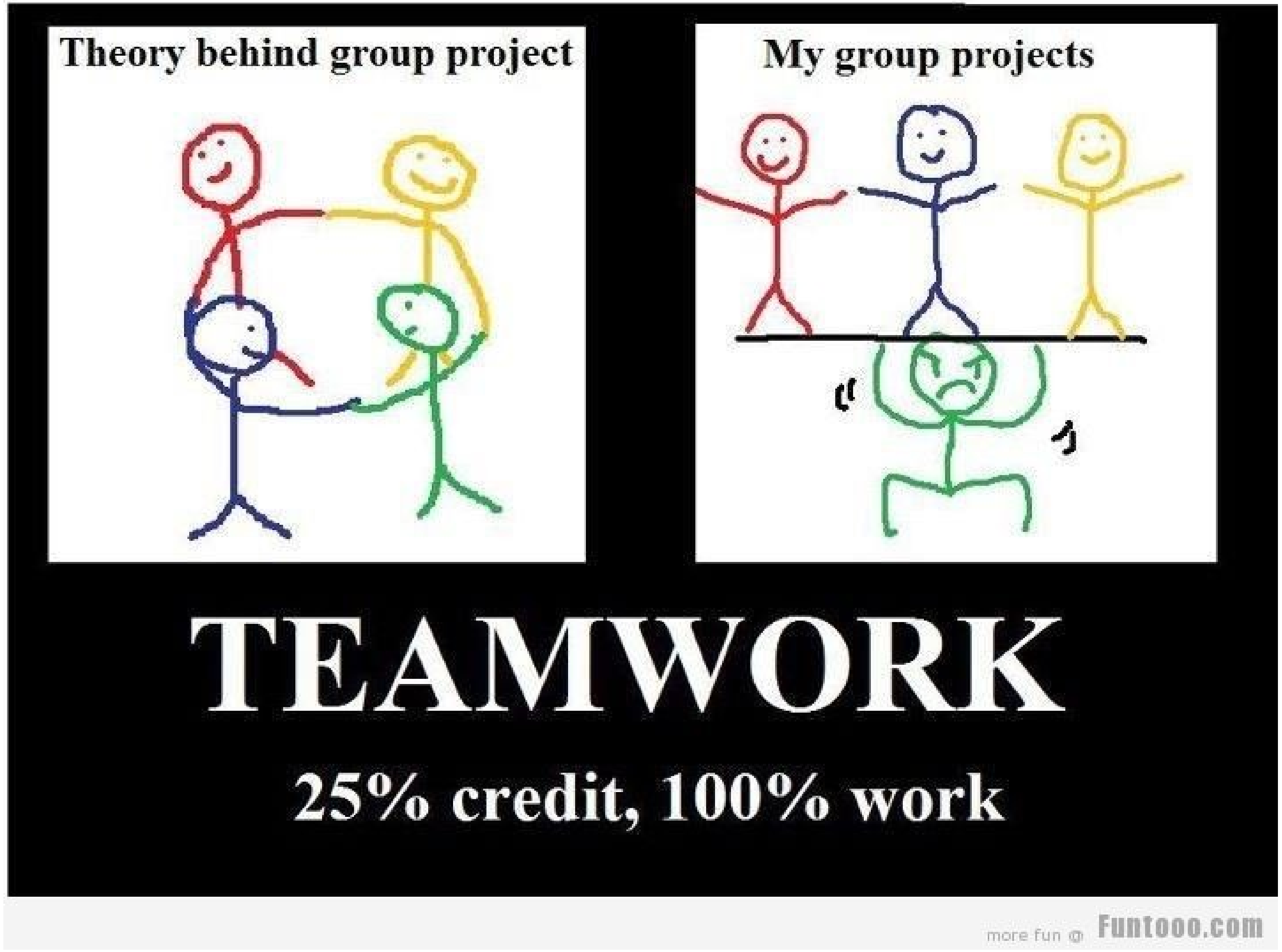
```
if((newUserName != _UserNa
 (newPassword != _Pass
 _UserName = newUser
 _Password = newP
 return true;
else {
```

If builders built buildings the way programmers wrote programs, then the first woodpecker that came along would destroy civilization.
Harry Weinberger

$\bar{x} = 54\%$

“The scores were all over the place. With the bottom 50% not getting it, this class is a perfect microcosm for the department as a whole.”

Teams


Theory behind group project
My group projects
TEAMWORK
25% credit, 100% work
more fun @ Funtooo.com

```
public Money add(Money money)
{
   int cents1 = 0, cents2 = 0, cents3 = 0;

   cents1 = this.getCentsTotal();
   if(this.isNegative())
      cents1 = -cents1;
   cents2 = money.getCentsTotal();
   if(money.isNegative())
      cents2 = -cents2;

   cents3 = cents1 + cents2;
   boolean positive = true;
   if(cents3 < 0)
   {
      positive = false;
      cents3 = Math.abs(cents3);
   }
   return new Money((cents3 / 100), (cents3 % 100), positive);
}
```

```
public Money add(final Money money)
{
   assert (money != null);

   return new Money(_cents + money._cents);
}
```

```
public int hashCode()
{
    int hash=0;
     try
        {
            MessageDigest sha = MessageDigest.getInstance("SHA-256");
            byte[] result =  sha.digest((this.toString()).getBytes());

            hash = ByteBuffer.wrap(result).getInt();
          }
          catch (NoSuchAlgorithmException ex)
          {
            System.err.println(ex);
          }
     return hash;
}
```

```
public int hashCode()
{
   return _cents;
}
```

```
public int compareTo(Money money){

    if( this.equals(money)==true)
        return 1;
    return 0;
}
```

```
public int compareTo(final Money money)
{
   assert (money != null);

   return (_cents - money._cents);
}
```

```
    public int compareTo(A_Currency money) {
        if(this.getValue().getCentsTotal() > money.getValue().getCentsTotal())
            return 1;
        if(this.getValue().getCentsTotal() < money.getValue().getCentsTotal())
            return -1;

        if(this.getDescription().compareTo(money.getDescription()) > 
money.getDescription().compareTo(this.getDescription()))
                return 1;
        else if(this.getDescription().compareTo(money.getDescription()) < 
money.getDescription().compareTo(this.getDescription()))
            return -1;
        else
            return 0;
    }

//since it isn't specified, I compare only on money unless they are equal,
  then I compare on description
```

```
 // Returns a new money as the result of subtracting a money to this one.
 public Money subtract(Money money){
     _centsTotal = _centsTotal - money.getCentsTotal();
     return this;
 }
```

```
public Money subtract(final Money money)
{
   assert (money != null);

   return new Money(_cents - money._cents);
}
```

```
public String toString()
{
   if(isPositive())
   {
      if(this.getCentsOfDollar() < 10) // "Dollar.0cent"
      {
         return "$" + this.getDollars() + ".0" + this.getCentsOfDollar();
      }
      else
      {
         return "$" + this.getDollars() + "." + this.getCentsOfDollar();
      }
   }
   else
   {
      if(this.getCentsOfDollar() < 10) // "Dollar.0cent"
      {
         return "$-" + this.getDollars() + ".0" + this.getCentsOfDollar();
      }
      else
      {
         return "$-" + this.getDollars() + "." + this.getCentsOfDollar();
      }
   }
}
```

```
public String toString()
{
   return String.format("$" + (isNegative() ? "-" : "") + "%d.%02d",
                        getDollars(), getCentsOfDollar());
}
```

```
public ArrayList<A_Currency> makeChange(Money amount)
{
   int amountDollars = amount.getDollars();
   int amountCents = amount.getCentsOfDollar();
   int profit = 0;
   while(amountDollars > 0)
   {
      if(amountDollars >= 10)
      {
         A_Currency ten = new CurrencyPaperDollar_10(amount);
         this.list.add(ten);
         amountDollars -= 10;
      }
      else if(amountDollars >= 5)
      {
        A_Currency five = new CurrencyPaperDollar_5(amount);
         this.list.add(five);
         amountDollars -= 5;
      }
      else if(amountDollars >= 2)
      {
         A_Currency two = new CurrencyPaperDollar_2(amount);
         this.list.add(two);
         amountDollars -= 2;
      }
      else if(amountDollars >= 1)
      {
         A_Currency one = new CurrencyPaperDollar_1(amount);
         this.list.add(one);
         amountDollars -= 1;

      }
   }
   while(amountCents >= 0)
   {
      if(amountCents >= 25)
      {
         A_Currency quarter = new CurrencyCoinCent_25(amount);
         this.list.add(quarter);
         amountCents -= 25;
      }
      else if(amountCents >= 10)
      {
        A_Currency dime = new CurrencyCoinCent_10(amount);
        this.list.add(dime);
        amountCents -= 10;
      }
      else if(amountCents >= 5)
      {
         A_Currency nickel = new CurrencyCoinCent_5(amount);
         this.list.add(nickel);
         amountCents -= 5;
      }
      else
      {
         profit = amountCents;
         amountCents -= profit;
         Money money = new Money(profit);
         //A_Currency toAdd = new A_CurrencyCoin(money, "Profit");
         //this.list.add(money);
      }
   }
   this.profits.add(profit);
   return this.list;
}
```

```
public Money(int dollars, int cents) {
    if(dollars > Integer.MAX_VALUE)
        throw new RuntimeException("passed value out of Integers range");
...

// Creates a money with a signed value in cents
public Money(int abs){
    value = Math.abs(cents);
    isPositive = true;
}

// Creates a money with a nonnegative value in dollars and cents
public Money(int dollars, int cents){
    value = Math.abs(cents) + Math.abs(dollars)*100;
    isPositive = true;
}
```

```
public java.lang.String getDescrption()
{
   return _description;
}


public CurrencyCoinCent_5()
{
   super(new Money(5), "nickle");
}
```

Delegating Work to Datatypes

```
private ComponentNavaidILSBeacon buildBeacon(final NavaidILSBeaconDescriptor descriptor)
{
   Assert.nonnull(descriptor);

   CoordinateWorld origin = getPosition().getCoordinateWorld();

   AngleNavigational azimuth = getAzimuth();

   Radius range = new Radius(descriptor.getDistance());

   CoordinatePolarNavigational bearing = new CoordinatePolarNavigational(azimuth, range);

   CoordinateWorld position2D = origin.resolveBearing(bearing);

   Altitude altitude = descriptor.getAltitude();

   CoordinateWorld3D position3D = new CoordinateWorld3D(position2D, altitude);

   String id = getID();

   VHFFrequency frequency = getFrequency();

   ComponentNavaidILSBeacon beacon = new ComponentNavaidILSBeacon(id, position3D, frequency);

   return beacon;
}
```

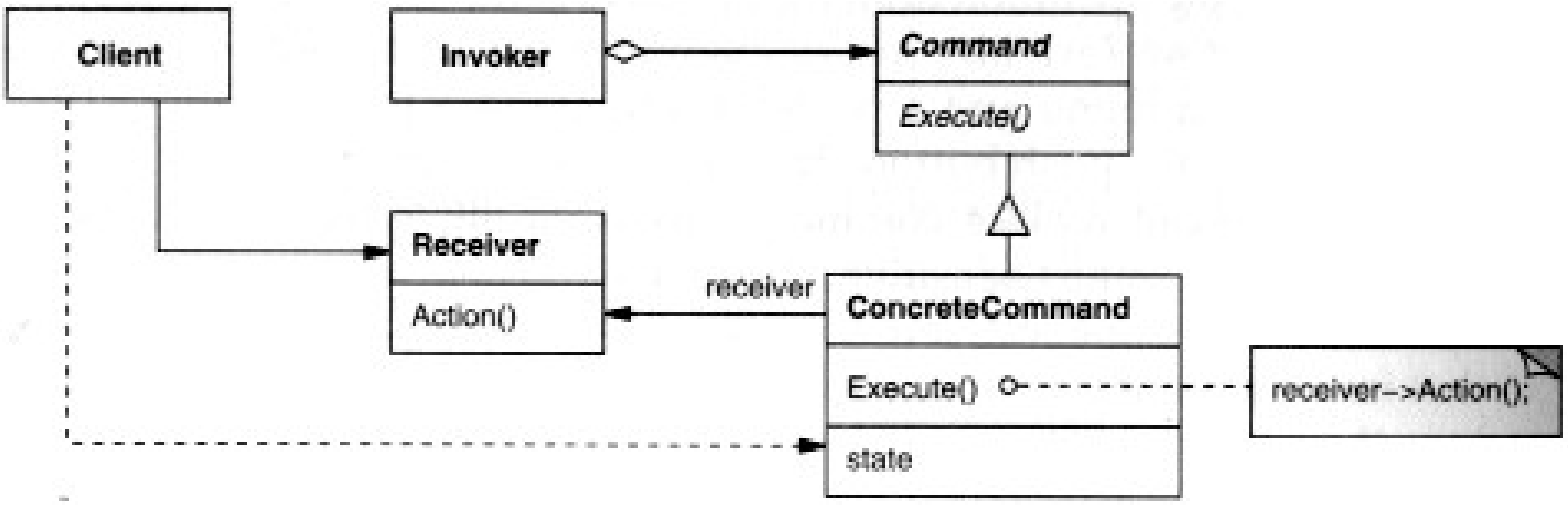
Delegating Work to Datatypes

```
public CoordinateWorld resolveBearing(final CoordinatePolarNavigational bearing)
{
   assert (bearing != null);

   CoordinateCartesianAbsolute origin = convertToCoordinateCartesianAbsolute();

   CoordinateCartesianAbsolute target = bearing.reciprocate().convertToCartesian(origin);

   Latitude latitude   = new Latitude(target.getY());
   Longitude longitude = new Longitude(target.getX());

   CoordinateWorld coordinate = new CoordinateWorld(latitude, longitude);

   return coordinate;
}
```

```
public CoordinateCartesianAbsolute convertToCartesian(final CoordinateCartesianAbsolute origin)
{
   assert (origin != null);

   double angleRadians = getAngle().convertToAngleMathematical().getValueRadians();

   double x = (  (Math.cos(angleRadians) * getRadius().getValue_()) + origin.getX());
   double y = (-((Math.sin(angleRadians) * getRadius().getValue_()) - origin.getY()));

   return new CoordinateCartesianAbsolute(x, y);
}
```

Command Pattern

- Encapsulates request as object, thereby allowing parameterizing of clients with different requests, queuing or logging requests, and supporting undoable operations
  - similar to message in Observer pattern
  - very useful for Interpreter pattern



Client
Invoker
Command
Execute()
Receiver
Action()
receiver
ConcreteCommand
Execute()
state
receiver->Action();