CSE 21
Intro to Computing II

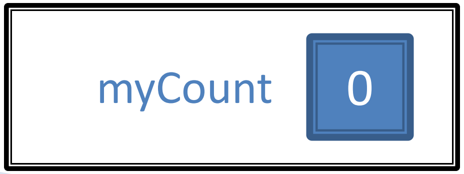
Lecture 9 – Inheritance (2)

Announcement

- Lab#8 due before start of next lab
  - Type your answers in a text file and submit it as an attachment
- Reading assignment
  - Chapter 7.11 to 7.14, 10.1 to 10.5 of textbook

Count Class Example

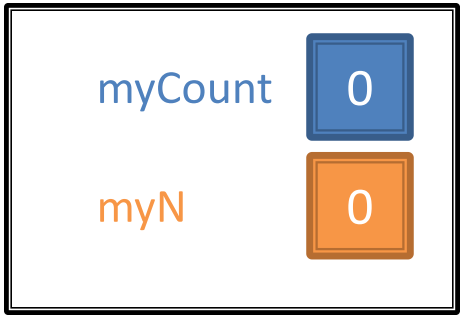
```
public class Counter {
 protected int myCount;
 public Counter() {
   myCount = 0;
 }
 public void increment(){
   myCount++;
 }
 public void reset() {
   myCount = 0;
 }
 public int value() {
   return myCount;
 }
}
```



myCount 0

```
public class ModNCounter
extends Counter {

   private int myN;
   public ModNCounter (int n){
     myN = n;
   }
   public int value ( ){
   // cycles from 0 to myN-1
    return myCount % myN;
   }
   public int max ( ){
     return myN-1;
   }
}
```



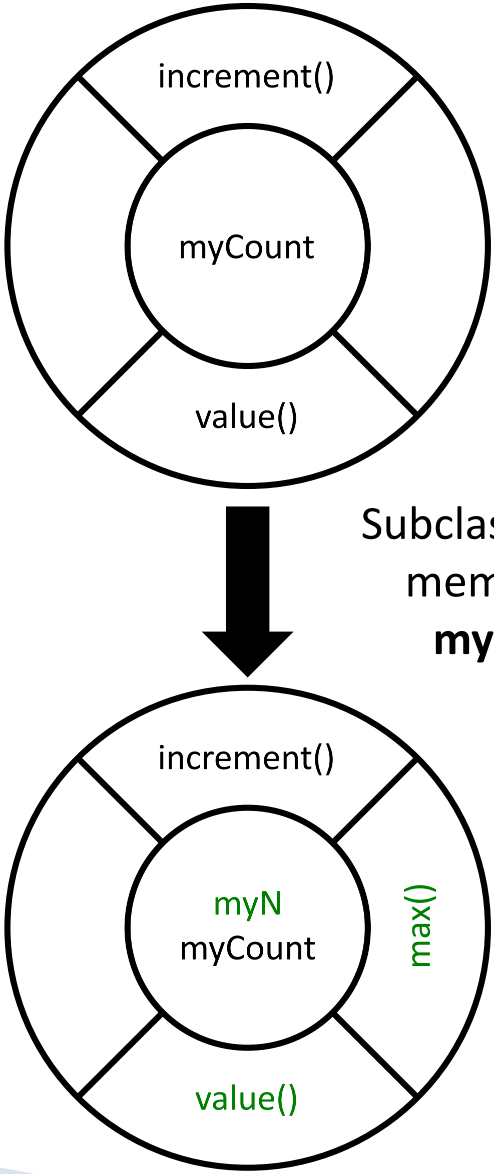
myCount 0

myN 0

Inheritance

Superclass

class Counter



Subclass inherits members of **myCount**

Subclass

class ModNCounter

Type Casting in Inheritance

- It will automatically Up-Convert type (int → double)
- Class types using inheritance follows the same rules
- Parent class is “higher” type than the child’s

```
Counter c = new ModNCounter(3); // legal (up)
ModNCounter mc = new Counter(); // not legal
ModNCounter mc = (ModNCounter) c; // legal (down, explicit)
```

- Anything you can do with a *Counter* you can also do with a *ModNCounter*
  - not vice versa

Type Checking

- It is OK to pass an object of one type to a method expecting another type that is a superclass.
- You get the version associated with the object, not the declared type.

```
ModNCounter mc = new ModNCounter(3);
Counter c = mc;
c.increment();
c.value(); // get the ModN version of value
```

- But you cannot call a method that may not exist:

```
c.max(); // illegal, because Counter does not have max()
```

- Why? Java is conservative

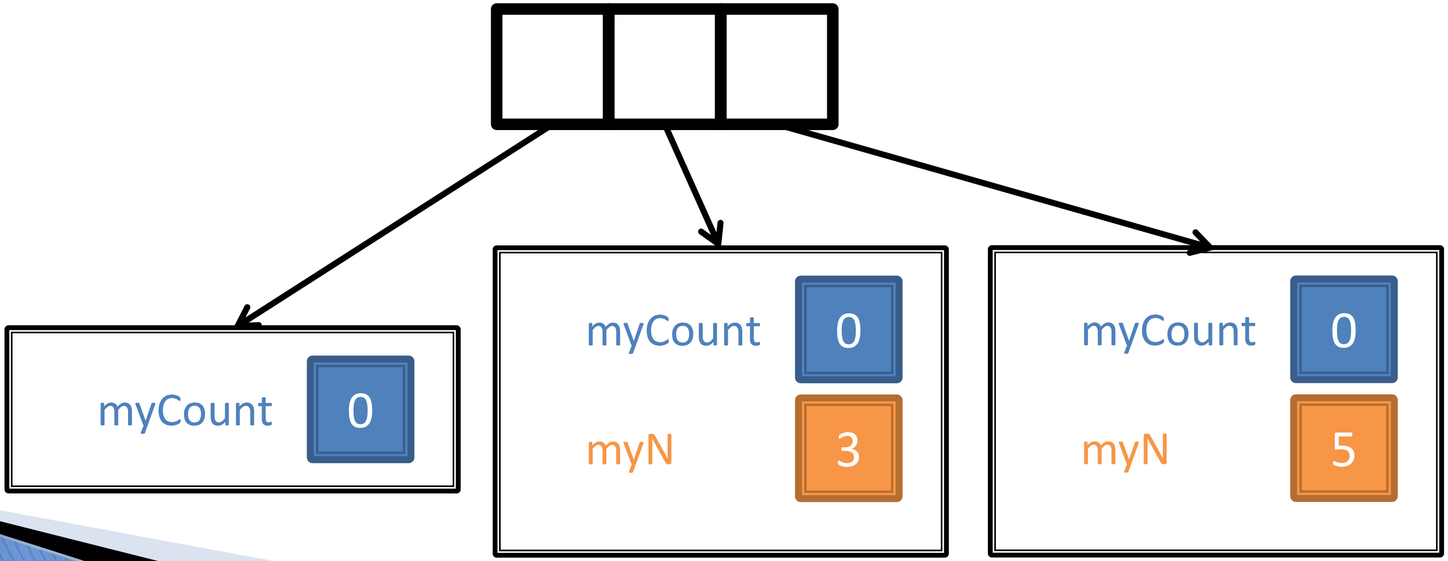
```
mc.max(); // OK, because mc is a ModNCounter
((ModNCounter)c).max(); // ERROR: because c may
                  // or may not be ModNCounter
```

Example

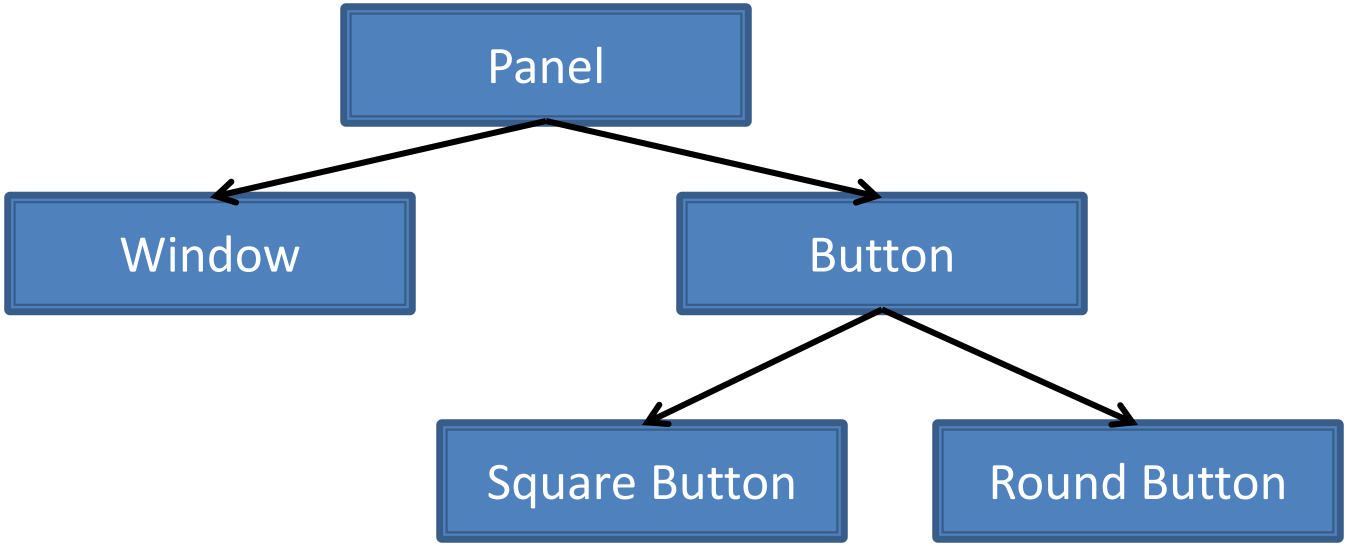
- Build an array of 3 Counters

```
Counter [] a = new Counter [3];
a[0] = new Counter();
a[1] = new ModNCounter(3);
a[2] = new ModNCounter(5);
```

Remember: need to use multiple "new" to create objects inside an array!



Inheritance Can be Multiple Levels

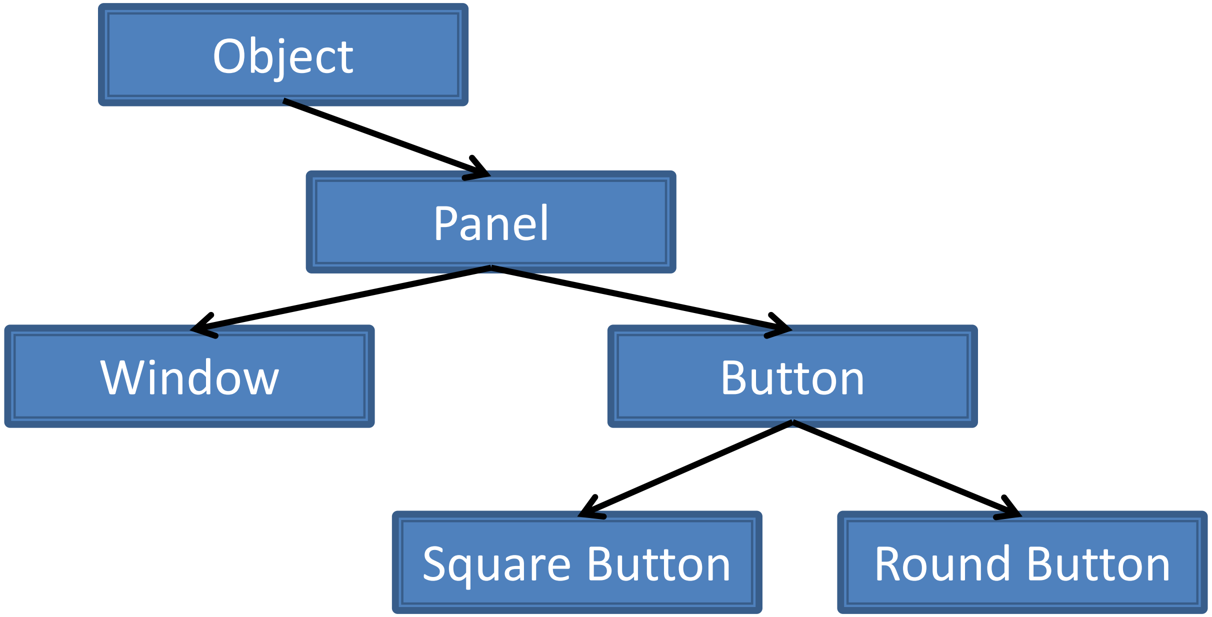


- Complex class hierarchies can be created

```
Panel [] p = new Panel [3];
p[0] = new Panel();
p[1] = new RoundButton();
p[2] = new Window();
```

Storing the graphic components of a program in an array

Object is at the Top in Java



- A the top of Java’s inheritance hierarchy is the special type ***Object***
- It comes with a few predefined methods such as ***toString***

Motivation: a Generic Search Algorithm

- We want a generic search algorithm to search for any kind of object in an array
- Class Object provides an ***equals()*** to test whether one object is equal to another
  - What does it mean when two objects are equal?
  - Simply checks if the 2 object references point to the same area of memory
    - Not very useful in practice
  - Compares the states of the 2 objects
    - Problem: different types of objects have different types of states
- We need to provide an ***equals()*** method in the class of the particular object type we are searching for
  - This is called ***Polymorphism***: a function that works on many types

Equals on Counters

- To check whether two *Counters* are equal:

Up cast to Counter type

```
public boolean equals (Object c) {
    return this.myCount == ((Counter) c).myCount;
} //Checks if myCounts are the same.
```

- Overriding equals for *ModNCounter*:

A new pointer pointing at the same object

```
public boolean equals (Object o) {
    ModNCounter mc = (ModNCounter) o;
    return (this.myCount == mc.myCount
        && this.myN == mc.myN);
} //Checks if myCounts and myN are the same.
```

A Search Algorithm

- This search code will work on any array of Objects
- As long as ***equals*** is properly defined

```
public class SearchAlg {
    public static int linearSearch(Object[] a, Object b) {
        int n = a.length;
        for (int i=0; i<n; i++) {
            if (a[i].equals(b))
                return 1;
        }
        return -1;
    }
}
```