CSE 21
Intro to Computing II

Lecture 8 – Inheritance (1)

Announcement

- Lab on Midterm Practice due before start of next lab
  - Type your answers in a text file and submit it as an attachment
  - No 2nd week re-submission
- Project #1
  - Due Friday (10/14) at 11:59PM
- Mid-term Exam on 10/19
  - During lecture (50 mins)
  - Open book/notes
  - Cover ch. 6 (Lecture 1 to beginning of Lecture 4)
- Reading assignment
  - Chapter 10.1 to 10.5 of textbook

Array of Objects

- Date johnny = new Date();
  - Creates an object pointed to by variable johnny
- Date[] birthdays = new Date[MAX];
  - Creates MAX # of Date pointers
  - Does not have objects yet
  - Not valid to use birthdays[0].setMonth(12) yet
  - It created MAX # of entries
- birthdays[0] = new Date(); // Now we can access
  - birthdays[0].setMonth(12);
- Need to instantiate two things for arrays (new)
  - Pointers using Square brackets
  - Objects using parenthesis

Arrays Usage

```
Date[] birthdays = new Date[MAX];

for (int i = 0; i < MAX; i++)
  birthdays[i] = new Date(2000 + i, i+1, i+15);

for (int i = 0; i < birthdays.length; i++)
  System.out.println(i +“ birthday is “ +
     birthdays[i].display());

for (int i = 0; i < MAX; i++)
  birthdays[i].setMonth( i+1 );

if (birthdays[5].getMonth() == 3)
  System.out.println(“Born in March”);
```

Object Parameters

- public void intro (Scanner input)
  - Takes in a Scanner object named input
- Date johnny = new Date();
  - Creates an object pointed to by variable/pointer johnny
- Date twin = johnny;
  - Points to the **SAME** object
- Date twin = new Date(johnny);
  - Creates a copy of the original object
    - Get the original value and put it in the new object
    - Different objects
  - public Date(Date original) {
    - this.setDay(original.getDay()); // this.day = original.getDay();
    - this.setMonth(original.getMonth()); // this month= original.getMonth();
    - this.setYear(original.getYear()); // this year = original.getYear();

Inheritance : Motivation

- Imagine you need an Object that is slightly different from the existing one
- Instead of re-designing an entire new object from scratch, you can inherit (or derive) the existing object and just “add” the needed modifications.
- Lets look at the Counter class
  - Counts how many times it’s been incremented (++)
  - Modulo_Counter inherits from Counter
    - Will reset myCount when it reaches a certain value
  - Call the new class ModNCounter

Count Class Example

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

```
public class ModNCounter
extends Counter {


}
```

ModNCounter c = new ModNCounter;
c.increment(); // THIS IS CORRECT

Count Class Example

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

```
public class ModNCounter
extends Counter {

  private int myN;

}
```

Additional instance variable

Count Class Example

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

```
public class ModNCounter
extends Counter {

  private int myN;
  public ModNCounter (int n){
    myN = n;
  }

}
```

Needs its own constructor

Count Class Example

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

```
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;
  }
}
```

Overriding (overloading) a method

Count Class Example

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

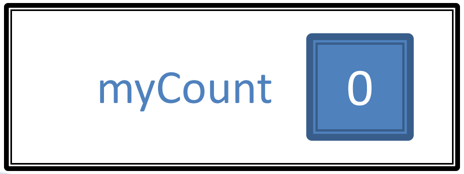
```
public class ModNCounter
extends Counter {

   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;
   }
}
```

New method

Count Class Example

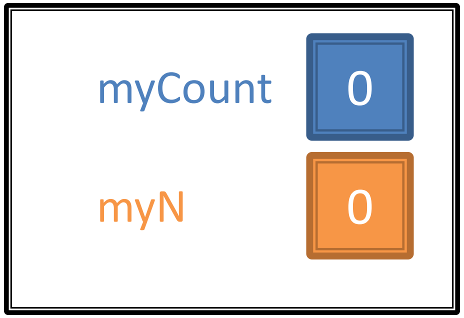
```
public class Counter {
 private int myCount;
 public Counter() {
   myCount = 0;
 }
 public void increment(){
   myCount++;
 }
 public void reset() {
   myCount = 0;
 }
 public int value() {
   return myCount;
 }
}
```



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

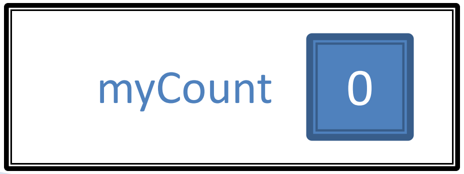
myN 0

Protected Access Specifier

- As written, *ModNCounter* will not compile !
- The *myCount* variable is private (only accessible in the *Counter* class)
- We can fix this by making it **protected**:
  - Only classes that “extend” *Counter* can access its protected variables/methods
- Three different Access types:
  - **public**: any class can read/modify
  - **protected**: only this class and subclass descendants can read/modify
  - **private**: only this class can read/modify

Count Class Example

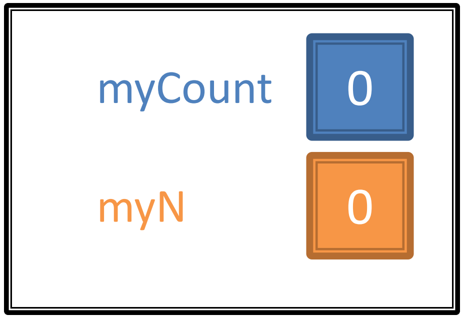
```
public class Counter {
 protected int myCount;
 public Counter() {
   myCount = 0;
 }
 public void increment(){
   myCount++;
 }
 public void reset() {
   myCount = 0;
 }
 public int value() {
   return myCount;
 }
}
```



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;
  }
}
```



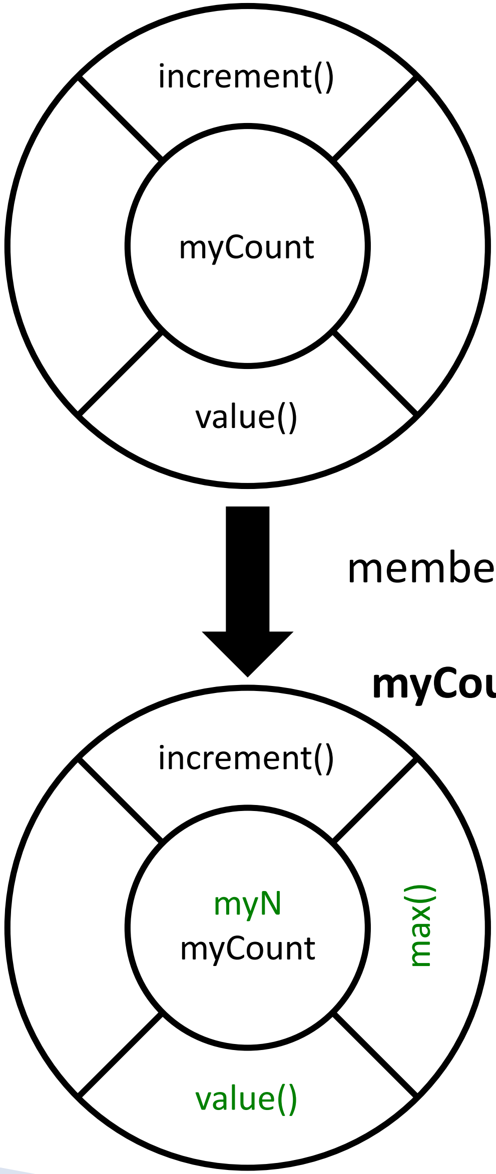
myCount 0

myN 0

Inheritance

Superclass

class Counter



Subclass inherits
members from superclass (public
or protected)
myCount, increment(), value()

Subclass

class ModNCounter

Inheritance Terminology

- Suppose class B inherits class A
- The classes form a part of a class hierarchy.
  - B is a subclass of A, B inherits A.
  - A is a superclass of B, A derives B.
  - The class immediately above a given class is known as its immediate superclass.
- A class inherits all (except private) members of the base class
  - Includes methods/variables inherited by that class
  - It can add additional variables and methods.
  - It can override (change) the inherited methods.
  - Can refer to super Class using keyword **super()**.

Relations

- Two common relationships are:
  - Is-a: All objects in one class also in another
    - E.g., a MyCounter is a Counter
  - Has-a: All objects in one class contain a reference to another object in another class
    - E.g., a Shop contains a Swiss Cheese
- Implement “Has-a” by adding objects as instance or class variables
- Implement “Is-a” by using inheritance
  - The new class is related to the class you inherit by an “is-a” relationship

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)
ModNCount mc = new Counter(); // not legal
ModNCount mc = (ModNCount) c; // legal (down)
```

- 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
```

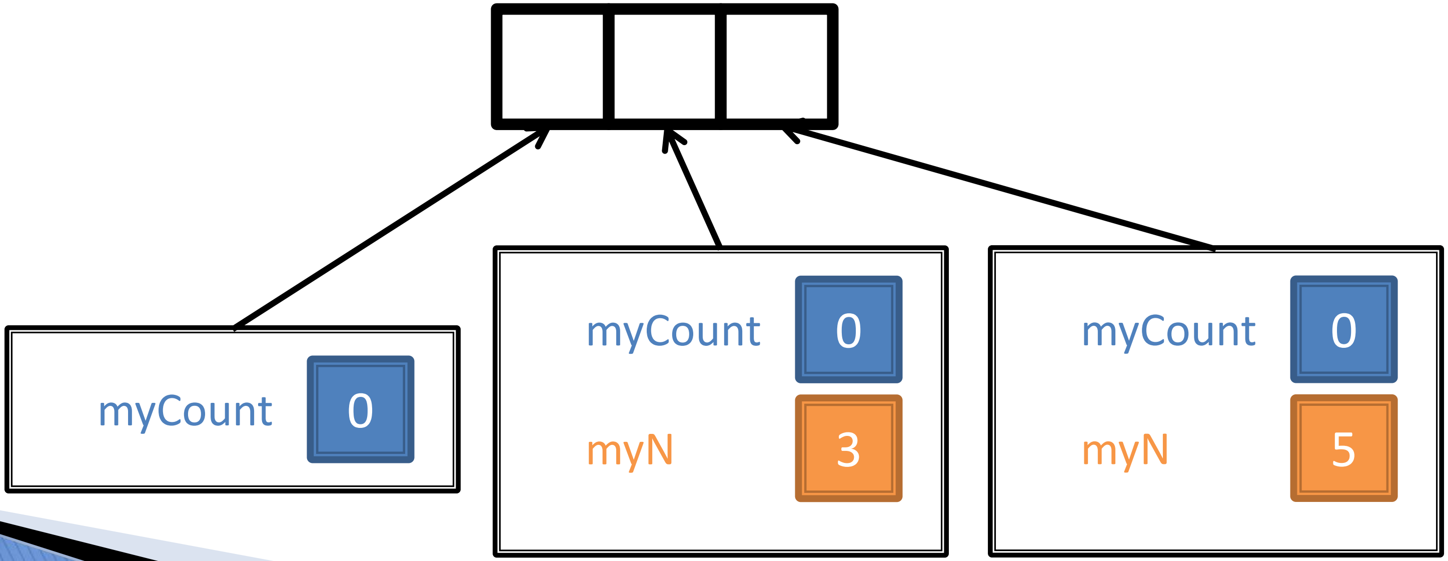
- Why? Java is conservative

```
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);
```


myCount
0
myCount
0
myN
3
myCount
0
myN
5