Introduction to Computer Science II

CS112-2012S-12

More Inheritance and Polymorphism

David Galles

Department of Computer Science
University of San Francisco

12-0: Inheritace

- Add “extends <classname>” to class defition
  - class Apartment extends Building { ... }
- Defines an “is-a” relationship
  - Apartment is a building
- Defines a superclass/subclass relationship
  - Building is the superclass
  - Aparment is the subclass

12-1: Inheritace

- Add “extends <classname>” to class defition
  - class Apartment extends Building { ... }
- Subclass inherits all of the methods / data from the superclass.
  - Examples from code

12-2: Inheritance

- Inheritance creates an is-a relationship

```
class Mammal extends Animal
{ ... }
class Cat extends Mammal
{ ... }
```

- A Mammal is an Animal
- A Cat is a Mammal, and is an Animal

12-3: Inheritance

- We can assign a subclass value to a superclass variable.

```
class Mammal extends Animal
{ ... }
class Cat extends Mammal
{ ... }

Animal a;
Mammal m = new Mammal();
a = m;
```

- All mamals are animals, so it makes sense that an animal a can be a mammal.

12-4: Inheritance

- We can pass a subclass value to a method that expects a superclass variable.

```
class Mammal extends Animal
{ ... }

boolean isPreditor(Animal a)
{ ... }

Mammal m = new Mammal();
isPreditor(m);
```

- All mamals are animals, so if we are expecting an animal, a mamal is OK

12-5: Inheritance

- We **cannot** assign a superclass value to a subclass variable

```
class Mammal extends Animal
{ ... }
class Cat extends Mammal
{ ... }

Animal a;
Mammal m = new Mammal();
m = a;
```

- Not all animals are mamals, so animal a might not be a mammal.

12-6: More on Inheritance

```
class A
{
   public int x;
   protected int y;
   private int z;
}

class B extends A
{
   public int w;
}

A c1 = new A();
B c2 = new B();
A c3 = new B();   /* Show memory contents */

c1.x = 3;  ?        c2.x = 3;  ?        c3.x = 1;   ?
c1.y = 4;  ?        c2.y = 4;  ?        c3.y = 2;   ?
c1.z = 5;  ?        c2.z = 5;  ?        c3.z = 3;   ?
c1.w = 6;  ?        c2.w = 6;  ?        c3.w = 4;   ?
```

12-7: More on Inheritance

```
class A
{
   public int x;
   protected int y;
   private int z;
}

class B extends A
{
   public int w;
}

A c1 = new A();
B c2 = new B();
A c3 = new B();   /* Show memory contents */

c1.x = 3; OK        c2.x = 3; OK        c3.x = 1; OK
c1.y = 4; NOT OK    c2.y = 4; NOT OK    c3.y = 2; NOT OK
c1.z = 5; NOT OK    c2.z = 5; NOT OK    c3.z = 3; NOT OK
c1.w = 6; NOT OK    c2.w = 6; OK        c3.w = 4; NOT OK
```

12-8: Yet More on Inheritance

```
class A
{
   public int x;
   protected int y;
   private int z;

   void set3(int a, int b, int c)
   {
       x = a;
       y = b;
       z = c;
   }
}

A c1 = new A();
A c2 = new B();

c1.set3(1, 2, 3);
c2.set3(4, 5, 6);
c2.set4(7, 8, 9, 10);
```

```
class B extends A
{
   privte int w;

   void set4(int a, int b, int c, int d)
   {
       x = a;  ?
       y = b;  ?
       z = c;  ?
       w = d;  ?
   }
}
```

12-9: Yet More on Inheritance

```
class A
{
   public int x;
   protected int y;
   private int z;

   void set3(int a, int b, int c)
   {
       x = a;
       y = b;
       z = c;
   }
}
```

```
class B extends A
{
   privte int w;

   void set4(int a, int b, int c, int d)
   {
      x = a;   OK
      y = b;   OK
      z = c;   BAD (private!)
      w = d;   OK
   }
}
```

```
A c1 = new A();
A c2 = new B();

c1.set3(1, 2, 3);
c2.set3(4, 5, 6);
c2.set4(7, 8, 9, 10);
```

How could we do this?

12-10: Yet More on Inheritance

```
class A
{
   public int x;
   protected int y;
   private int z;

   void set3(int a, int b, int c)
   {
       x = a;
       y = b;
       z = c;
   }
}

A c1 = new A();
A c2 = new B();

c1.set3(1, 2, 3);
c2.set3(4, 5, 6);
c2.set4(7, 8, 9, 10);
```

```
class B extends A
{
   privte int w;

   void set4(int a, int b, int c, int d)
   {
      set3(a, b, c)
      w = d;
   }


}
```

12-11: Yet More on Inheritance

```
class A
{
   public int x;
}

class B extends A
{
   public int d;
}

In main:
--------
A a;
B b;

a.x = 3;  OK?
b.x = 4;  OK?
b.d = 5;  OK?
```

12-12: Yet More on Inheritance

```
class A
{
   public int x;
}

class B extends A
{
   public int d;
}

In main:
--------
A a;
B b;

a.x = 3;  NOT OK -- cannot use class variable until "new" is called
          All class variables (and arrays!) are stored on the heap
b.x = 4;  NOT OK
b.d = 5;  NOT OK
```

12-13: Yet More on Inheritance

```
class A
{
   public int x;
   public C c;
   public int intArray[];
   public C[] cArray;
}

class B extends A
{
   public int d;
}

class C
{
   public int e;
}
```

```
In Main
--------
A a = new A();
B b = new B();

a.x = 3;              OK?
a.c.e = 4;            OK?
b.x = 5;              OK?
b.d = 6;              OK?
a.intArray[2] = 4;    OK?
```

12-14: Yet More on Inheritance

```
class A
{
   public int x;
   public C c;
   public int intArray[];
   public C[] cArray;
}

class B extends A
{
   public int d;
}

class C
{
   public int e;
}
```

```
In Main
--------
A a = new A();
B b = new B();

a.x = 3;              OK
a.c.e = 4;            NOT OK
b.x = 5;              OK
b.d = 6;              OK
a.intArray[2] = 4;    NOT OK
```

12-15: Yet More on Inheritance

```
class A
{
   public int x;
   public C c;
   public int intArray[];
   public C[] cArray;
}

class B extends A
{
   public int d;
}

class C
{
   public int e;
}
```

```
In Main
--------
A a = new A();
A a2 = new A();

a.cArray = new C[5];          OK?
a.cArray[2].e = 5;            OK?
a2.e = 4;                     OK?
a2.intArray = new int[5];     OK?
a2.intArray[2] = 4;           OK?
```

12-16: Yet More on Inheritance

```
class A
{
   public int x;
   public C c;
   public int intArray[];
   public C[] cArray;
}

class B extends A
{
   public int d;
}

class C
{
   public int e;
}
```

```
In Main
--------
A a = new A();
A a2 = new A();

a.cArray = new C[5];          OK
a.cArray[2].e = 5;            NOT OK
a2.e = 4;                     NOT OK
a2.intArray = new int[5];     OK
a2.intArray[2] = 4;           OK
```

12-17: Yet More on Inheritance

```
class A
{
   public int x;
   public C c;
   public int intArray[];
   public C[] cArray;
}

class B extends A
{
   public int d;
}

class C
{
   public int e;
}
```

```
In Main
--------
A a[] = new A[2];
int x;
int y[] = new int[3];

a[2].x = 4                      OK?
a[2].cArray = new C[3];         OK?
a[2].cArray[2].e = 4            OK?
a.x = 3                         OK?
x = 3;                          OK?
y[3] = 4;                       OK?
```

12-18: Yet More on Inheritance

```
class A
{
   public int x;
   public C c;
   public int intArray[];
   public C[] cArray;
}

class B extends A
{
   public int d;
}

class C
{
   public int e;
}
```

```
In Main
--------
A a[] = new A[2];
int x;
int y[] = new int[3];

a[2].x = 4                      NOT OK
a[2].cArray = new C[3];         NOT OK
a[2].cArray[2].e = 4            NOT OK
a.x = 3                         NOT OK
x = 3;                          OK
y[3] = 4;                       OK
```

12-19: Yet More on Inheritance

```
class A
{
   public int x;
   public C c;
   public int intArray[];
   public C[] cArray;
}

class B extends A
{
   public int d;
}

class C
{
   public int e;
}
```

```
In Main
--------
A a[] = new A[3];

a[0] = new B();     OK?
a[1] = new A();     OK?
a[2] = new B();     OK?
a[3] = new A();     OK?
a[0].x = 2;         OK?
a[a[0].x].x = 4     OK?
a[2].e = 5;         OK?
a[2].c = new C();   OK?
a[2].c.e = 6;       OK?
```

12-20: Yet More on Inheritance

```
class A
{
   public int x;
   public C c;
   public int intArray[];
   public C[] cArray;
}

class B extends A
{
   public int d;
}

class C
{
   public int e;
}
```

```
In Main
--------
A a[] = new A[3];

a[0] = new B();     OK
a[1] = new A();     OK
a[2] = new B();     OK
a[3] = new A();     OK
a[0].x = 2;         OK
a[a[0].x].x = 4     OK
a[2].e = 5;         NOT OK
a[2].c = new C();   OK
a[2].c.e = 6;       OK
```

12-21: Object

- Every class in java is a subclass of Object
- If a class has no “extends”, it is assumed to extend Object directly

```
class myClass
{
...
}
```

is the same as:

```
class myClass extends Object
{
...
}
```

12-22: Object Heirarchy

```
class Animal
{
...
}
class Mammal extends Animal
{
...
}
class Reptile extends Animal
{
...
}
class Cat extends Mammal
{
...
}
class Dog extends Mammal
{
...
}
```

```
class Vehicle
{
...
}
class Car extends Vehicle
{
...
}
class Bicycle extends Vehicle
{
...
}
class HybridCar extends Car
{
...
}
```

12-23: Object

- In Java, *everything* that is allocated on the heap (arrays, Strings, all classes – everything except int, boolean, float, double, etc), is a subclass of Object

```
String s = "Hello!"     <-- Implicit call to new!
int array[] = new int[10];
MyClass c;

Object o;
o = s;
o = array;
o = c;
```

12-24: Object

- “Object” class is really useful for containers
  - For Project 2, created an arraylist of strings
  - What if you wanted an array of something else – integers, booleans, some other class that you defined??
  - We could create several different kinds of lists – but is there a better way?

12-25: Containers

- Create a list of $Objects$ instead of a list of Strings, ints, etc.
- This list can store any Object – and every class is an object.

12-26: ArrayList class

```
class ArrayList {
   public ArrayList() { ... }
   boolean add(Object obj) { ... }
   void add(int index, Object obj { ... }
   void clear() { ... }
   Object remove(int index) { ... }
   Object get(int index) { ... }
   int indexOf(Object obj) { ... }
   int size() { ... }
   // Other methods too ..
}
```

```
ArrayList L = new ArrayList();
String s1 = "hello";
String s2 = "world";
L.add(s1);
L.add(s2);
```

```
ArrayList L2 = new ArrayList();
C c1 = new C();
C c2 = new C();
L2.add(c1);
L2.add(c2);
```

12-27: Wrapper Classes

- There are times when we want primitives (int, boolean, etc) to behave like “regualr” objects. (For instace, if we want to create an ArrayList of ints)
- We can use the “Wrapper classes” Integer, Boolean, etc.
- Wrapper classes are essentially just classes with a single instance variable, representing the wrapped value

12-28: Wrapper Clases

```
class Integer
{
    private int value;

    public Integer(val)
    {
        value = val;
    }

    public int intValue()
    {
        return value;
    }

    // Other methods ...
}
```

12-29: Wrapper Clases

- All wrapper classes (like *all* classes) are subclassed off Object

```
Integer i1 = new Integer(3);
Integer i2 = new Integer(4);

Object o;
o = i1;
o = i2;
```

12-30: Wrapper Classes & Containers

```
ArrayList L1 = new ArrayList();
ArrayList L2 = new ArrayList();

L1.add(new Integer(3));       L2.add(new Double(3.14159);
L1.add(new Integer(3));       L2.add(new Double(2.71828);
L1.add(new Integer(4));       L2.add(new Double(1.61803);
```

12-31: Wrap. Classes & Containers

```
ArrayList L1 = new ArrayList();
ArrayList L2 = new ArrayList();
ArrayList L3 = new ArrayList();

L1.add(new Integer(3));       L2.add(new Double(3.14159);
L1.add(new Integer(3));       L2.add(new Double(2.71828);
L1.add(new Integer(4));       L2.add(new Double(1.61803);

L3.add(new Integer(3));
L3.add(new Double(2.9176));
L3.add("foobar");
```

12-32: Object

- Technically, primative types are not objects (stored on the stack, not the heap
- Need to use wrappers to make them behave like objects

```
int myInt = 7;

Object o1, o2;
o1 = new Integer(myInt);  // OK!
o2 = myInt;               // Primatives not Objects!  but ...
```

12-33: Object

- Technically, primative types are not objects (stored on the stack, not the heap
- Need to use wrappers to make them behave like objects

```
int myInt = 7;

Object o1, o2;
o1 = new Integer(myInt);  // OK!
o2 = myInt;               // Primatives not Objects!  but ...
                          // Java compiler automatically wraps this for you

(changed to o2 = new Integer(myInt) by compiler behind the scenes)
```

12-34: Polymorphism

- OK, so we have an array of Objects ...
- What can we do with it?
  - Print out the object
  - Convert it to string (using toString)

How can we do something more useful?

12-35: Polymorphism

- We can “override” methods described in a superclass
  - Create a method in the subclass with the same “signature” as the superclass
    - Same name, same number and type of parameters
  - Subclass will use the new definition of the method

12-36: Polymorphism

```
class A
{
   void print()
   {
      System.out.println("Hellow from A");
   }
}
class B extends A
{
   void print()
   {
      System.out.println("Hellow from B");
   }
}

A classA = new A();
B classB = new B();

classA.print();
classB.print();
```

12-37: Polymorphism

```
class A
{
   void print()
   {
      System.out.println("Hellow from A");
   }
}
class B extends A
{
   void print()
   {
      System.out.println("Hellow from B");
   }
}

A classA = new B();

classA.print();
```

12-38: Polymorphism

- Superclass contains a method “Print”
- Subclass overrides the method “Print”
- Assign a subclass value to a superclass variable
- Call the print method of the superclass variable
  - Uses the subclass version

12-39: Polymorphism

- We’ve actually seen this before
  - toString()

12-40: Polymorphism

```
class A
{
   void print()
   {
      System.out.println("Hellow from A");
   }
}

class B extends A
{
   void print()
   {
      System.out.println("Hellow from B");
   }
}
```

```
In main:
---------
A aArray = new A[5];
aArray[0] = new A();
aArray[1] = new A();
aArray[2] = new B();
aArray[3] = new B();
aArray[4] = new A();

for (int i = 0; i < 5; i++)
{
   aArray[i].print();
}
```

12-41: Polymorphism

```
class A {
   void print() {
      System.out.println("Hello from A")
   }
}
class B extends A {
   void print() {
      System.out.println("Hello from B")
   }
}
class C extends A {
   void print() {
      System.out.println("Hello from C")
   }
}
class D extends B {
   void print() {
      System.out.println("Hello from D")
   }
}
```

```
In main:
--------
A aArray = new A[5];
aArray[0] = new B();
aArray[1] = new C();
aArray[2] = new D();
aArray[3] = new C();
aArray[4] = new D();

for (int i = 0; i < 5; i++)
{
   aArray[i].print();
}
```

12-42: MiniLab

- Create a superclass Operator, which contains a single function: `int doOperation(int operand1, int operand2)` which returns the first operand
- Create four subclasses of Operator: Add, Subtract, Multiply, and Divide. These sublclasses should override the doOperation method to either add, subtract, multiply, or divide the operands
- Create a main program that allocates an array of length 4 of type Operator
- Assign each element of the array a different new operator instance (add,subtract,multiply,divide)
- Calculate opArray[i].doOperation(8, 3) for i = 0 .. 3
- Use the opArray to calcuate the sum and product