CSE115 / CSE503
Introduction to Computer Science I

Dr. Carl Alphonce

343 Davis Hall

alphonce@buffalo.edu

Office hours:

Tuesday 10:00 AM – 12:00 PM*

Wednesday 4:00 PM – 5:00 PM

Friday 11:00 AM – 12:00 PM

OR request appointment via e-mail

**Tuesday adjustments: 11:00 AM – 1:00 PM on 10/11, 11/1 and 12/6*

ROADMAP

Last time

Inheritance

Today

Inheritance

int representation in detail

Coming up

floating point representation

search

ANNOUNCEMENTS

EXAM 2 NOTICE

DATE: Tuesday November 15

TIME: 8:45 PM – 9:45 PM

LOCATION: as for exam 1

COVERAGE:

lecture material from 9/26 up to and including 11/04
lab material labs 4 – lab 9

readings: all assigned up to and including 13.4

HAVE A CONFLICT? ***Let me know by tonight***

Send e-mail with PDF of documentation.

Subject line: [CSE115] Exam 2 conflict documentation

BRING: your UB card

NO ELECTRONICS: cell phone, calculator, etc.

REVIEW

inheritance

interface-interface and class-class inheritance

the class Object

single inheritance

inheritance/partial overriding/total overriding

super

INHERITANCE

overriding summary

total (complete) overriding

a subclass provides an entirely new definition for a method which would otherwise have been inherited from the superclass

partial overriding

a subclass provides a definition for a method which would otherwise have been inherited from the superclass, but calls the superclass version via super.

inheritance

a subclass does not provide an alternate defintion for a method defined in the superclass, which is inherited.

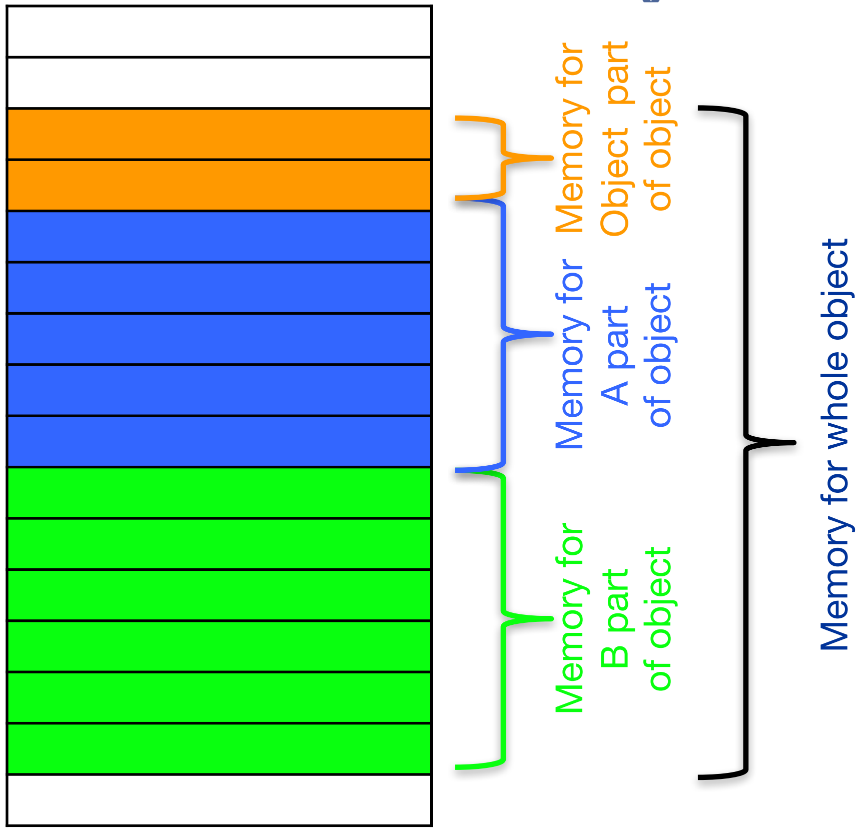
object layout in memory

When a class is instantiated, memory is reserved for the whole object, including parts contributed by ancestor classes.

CODE

```
public class A {
  private int _x;
  public A(int x) {
    _x = x;
} }
public class B extends A {
  private double _y;
  public B(double y) {
    _y = y;
} }
```

MEMORY for new B()


Memory for Object part of object
Memory for A part of object
Memory for B part of object
Memory for whole object

```
public class A {
  private int _x;
  public A(int x) {
    _x = x;
  }
}

public class B extends A {
  private double _y;
  private boolean _z;
  public B() {
    this(1.0, false);
  }
  public B(double d, boolean b) {
    super(5);
    _y = d;
    _z = b;
  }
}
```

When creating an instance of B, we have a choice of which constructor to use.

Using the second constructor we can specify the initial values of the instance variables by passing arguments.

This constructor calls the superclass's constructor.

```
public class A {
  private int _x;
  public A(int x) {
    _x = x;
  }
}

public class B extends A {
  private double _y;
  private boolean _z;
  public B() {
    this(1.0, false);
  }
  public B(double d, boolean b) {
    super(5);
    _y = d;
    _z = b;
  }
}
```

When creating an instance of B, we have a choice of which constructor to use.

Using the first constructor “default” values are provided for the two instance variables; their values are set by the second constructor, which is called from the first (the ‘this(1.0,false)’ call).

The second constructor explicitly calls the superclass constructor with argument 5 (the super(5) call).

overloading

Defining more than one constructor for a class is an example of *overloading*

In general, a *name* can be overloaded with multiple definitions, as long as the correct interpretation of the name can be determined by the compiler from context.

Methods/constructors can be overloaded as long as the name can be disambiguated based on the call.

For methods/constructors disambiguation is carried out based on the number, type and order of parameters.

For example, you cannot define two methods with the same name and the same parameter lists.

The return type is not considered when trying to disambiguate a call.

default constructor

If no explicit constructor is defined for a class, the compiler provides one.

This “default” constructor takes no arguments (i.e. it has an empty parameter list) and an empty body.

default constructor

```
public class A {
}
```

default constructor

```
public class A {
    public A() {
   }
}
```

default constructor

```
public class A {
    public A() {
        super();
    }
}
```

constructor chaining

Any constructor which does not explicitly call a superclass constructor implicitly invokes the no-argument constructor of the superclass.

An explicit invocation of a superclass constructor is done using ‘super’.

The first statement in a constructor must be a call to a constructor. Often this call is to a superclass constructor, but it can be to another constructor of the same class. We’ll see an example a few slides from now.

The first statement of a constructor body may be an explicit invocation of another constructor of the same class or of the direct superclass (§8.8.7.1).

If a constructor body does not begin with an explicit constructor invocation and the constructor being declared is not part of the primordial class Object, then the constructor body implicitly begins with a superclass constructor invocation "super();", an invocation of the constructor of its direct superclass that takes no arguments.

https://docs.oracle.com/javase/specs/jls/se8/html/jls-8.html#jls-8.8.7.1

subtle errors

The compiler injects a default constructor into a class definition ONLY if there is no explicit constructor defined.

Therefore, defining an explicit constructor with a non-empty parameter list in superclass which previously had a default constructor will cause errors in subclass constructors (since they rely on a call (implicit or explicit) to a no-argument/default constructor, which no longer exists)

Eclipse demonstration

See code in lecture code project in student repository.