CSE115 / CSE503
Introduction to Computer Science I

Dr. Carl Alphonce
343 Davis Hall
alphonce@buffalo.edu

Office hours:
Thursday 12:00 PM – 2:00 PM
Friday 8:30 AM – 10:30 AM
OR request appointment via e-mail
Turn off and put away electronics:

- cell phones
- pagers
- laptops
- tablets
- etc.

© Dr. Carl Alphonce
Today

inheritance and constructor chaining

Coming up

primitive types in more detail
linear and binary search
FINAL EXAM SCHEDULE

Arranged by the university
Check HUB for date/time/room

UNCLAIMED EXAMS

Pick up at end of class today
Inheritance
object layout in memory
When a class is instantiated, memory is reserved for the whole object, including parts contributed by ancestor classes.

public class A {
    private int _x;
    public A() {
        _x = 3;
    }
}

public class B extends A {
    private double _y;
    public B(double y) {
        _y = y;
    }
}
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.
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).
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.
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.
public class A {
}

default constructor
public class A {
    public A() {
    }
}
public class A {
    public A() {
        super();
    }
}
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.”
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)
Primitive Types
Java has eight primitive types

- boolean
- integral types:
  - signed: long, int, short, byte
  - unsigned: char
- floating point types: double, float

Values of the primitive types are not objects
- no properties
- no capabilities
values: 0, 1, -1, 2, -2, ...
maximum int: 2147483647 = +2^{(32-1)}-1
minimum int: -2147483648 = -2^{(32-1)}
operations: + - * / %

5 + 2 = 7  
5 - 2 = 3  
5 * 2 = 10 
5 / 2 = 2 (quotient)  
5 % 2 = 1 (remainder)
representation used differs according to whether type is signed (byte, short, int, long) or unsigned (char):

- signed integral values are represented using “two’s complement” representation
- unsigned integral values are represented using “binary” representation

size of representation differs according to type:

- byte is 1 byte wide (1 byte = 8 bits)
- short is 2 bytes wide
- int is 4 bytes wide
- long is 8 bytes wide

main point: values of different types have different representations – you can’t “mix and match”!
Notice that all of these operators take two int arguments, and produce an int result.

There is hardware circuitry to perform these operations.
Two's complement

uses a fixed-width encoding
encodes a limited range of values
encodes both negative and non-negative values

familiar properties hold

✓ unique representation of zero (0 = -0)
✓ \(x + 0 = 0 + x = x\)
✓ \(x = -(-x)\)
✓ \(x + (-x) = 0\)
✓ \(x - y = x + (-y)\)

this last property lets us use addition circuitry to perform subtraction (to subtract \(y\) from \(x\), negate \(y\) and add to \(x\))