CMSC 131

Object-Oriented Programming I

Overloading

Dept of Computer Science
University of Maryland College Park

This material is based on material provided by Ben Bederson, Bonnie Dorr, Fawzi Emad, David Mount, Jan Plane

Method Overloading

- **Overloading**:
  - Java allows methods to have the **same name**, even within the same class.
  - Where have we seen overloading before? Constructors!
  - For example, given a Date class we can add the following methods to change the current date. (implementations have been omitted)

```
public void setDate( int m, int d, int y ) { ... }      // month given as integer
public void setDate( String m, int d, int y ) { ... }   // month given as string
public void setDate( int m, int y ) { ... }             // day defaults to 1
```

- **Sample calls**:

```
Date dueDate = new Date( 10, 5, 2004 );     // set initial due date
dueDate.setDate( 10, 7, 2004 );             // delay the due date
dueDate.setDate( "Nov", 12, 2004 );         // delay it further
dueDate.setDate( 1, 2005 );                 // delay until next year
```

- **Question**: How does Java know which one to call?
- **Answer**: It looks at the number and of types of arguments

Method Overloading and Signatures

- **Overloading**: using the **same identifier name** for **different methods**. Usually these methods perform very similar functions, but we want to provide different ways of accessing it (for convenience of the class user). Java determines which one to call based on the method's **signature**.
- **Signature**: of a method consists of the name of the method and the types of the parameters.

Example:

```
public float doSomething( int x, double z, double w, String s )
```

Corresponding Signature:

```
doSomething( int, double, double, String )
```

Prototype: of a method is the signature of the method with a return type and any additional modifiers

Method Overloading and Signatures (continued)

- Note that the **return type** of a method is **not** part of the signature. You cannot overload two methods with the same parameter types but different return types.

- **Example**: Two methods convert temperature in Fahrenheit to Celsius. One rounds to the nearest integer, the other doesn't.

```
public int toCelsius( double t ) { ... }
public double toCelsius( double t ) { ... }
...
System.out.println( toCelsius( 98.6 ) );
```

Cannot resolve this based on the parameter types alone.

- **Which method should be called?** Unfortunately, Java cannot read your mind. (Solution: Use different names.)

Parameter Type Promotion

- **Automatic Casting**: We have seen that, in assignment statements, Java automatically promotes numeric types to the higher type:

```
int total = … ;
double average = total;
```

total's value is promoted to a double before doing the assignment.

- **Promotion of Parameters**: In the same way, Java automatically promotes each argument to match the type of its formal parameter. Math.sqrt( ) expects an argument of type **double**.

```
int area = 1024;
double s = Math.sqrt( area );
```

area's value is promoted to a double before passing it as a parameter.

Ambiguous Overloading

- Because of type promotion, there are times when Java cannot figure out which method to call.

```
public void fooBar( int x, double y ) { … }
public void fooBar( double u, int v ) { … }
…
fooBar( 10, 23.0 );// okay, use the first
fooBar( 10.0, 23 );// okay, use the second
fooBar( 10, 23 );              // ???
```

Depending on the call, this setup might be ambiguous

- Do we promote 23 to 23.0 and call the first, or promote the 10 to 10.0 and call the second?
- Java issues a **compile-time error**, since it cannot resolve the ambiguity
- How can we solve the problem?