

Developing Object-Oriented Software

more method overloading
packages
Generics

Widening

- Method overloading is simple when it comes to the number of arguments
- But what about argument types?
- For example, if you have two byte arguments, and there is a method which accepts floats and one with the same name which accepts ints, which method is called?

Widening

- Java uses widening to figure it out.
- Remember widening is modifying the variable type to one with a higher level of precision.
- `byte > short > int > long > float > double`
- Java will try widening the arguments, and use the method which requires the least widening

Widening

- The number of arguments which must be widened is also taken into account - Java will call the method which requires the fewest arguments to be widened

packages

- Let's imagine you have a very large Java project
- You want to use Node as a classname, but someone else in the project has already has a class named Node

packages

- The solution is putting them in different packages, via the package keyword. This puts them in two separate “namespaces”. In the same package, you can refer to classes without the package name. Outside of that package, you need to import it first.
- Examples (go above class, after import statements):
- `package com.laboon.project.Node;`
- `package com.wallawalla.Node;`

packages

- Now somebody working in package.com.jablonsky who wants to use Node must import Node from either com.laboon.project or com.wallawalla.Node;
- Note package naming - think of them as directories and subdirectories. In fact, these are how they should be stored on disk!
- You can get all the classes from a package with .* or just one by naming it specifically.

importing classes from packages

- You've already done this!
- ```
import com.laboone.project.Node;
import java.util.*;
import java.io.FileReader;
```




# Running packaged code

- java expects that if you run packaged code, it's in the directory structure you mention in the package. For example, `com.cat.Kitty` should be in `./com/cat/Kitty.class`
- If java says it can't find the class, this is probably why! (I make this mistake myself often)

# default package

- If you don't say what package a class is in, it goes in the default package (this is what you have been doing the entire class)
- Remember that protected variables can also be accessed by classes of the same package, so if you package your code up nicely into packages, you can avoid some of the data sharing inherent in protected

# Generics

- Professor Laboon  instanceof, but there are some problems with our relationship:
  - Run-time checks < Compile-time checks
  - Manual checks for correct data type
  - What if you forget a class? Have to remember all the ones you want to deal with
  - Nothing stopping people from putting the wrong things INTO your ArrayLists, etc.



# Generics

- Brand-new (as of Java 1.5)
- Generics allow you to specify the class you want a Collection (or other kind of object) to store
- Helps with code re-use, compile-time checks, input and output verification, etc.
- Sorry, instanceof, but it's time for me to move on..



# Generics

- Example
  - `ArrayList<String> a = new ArrayList<String>();`
    - Can ONLY store Strings - can check at compile time
    - Everything that comes out is guaranteed to be a string
    - No explicit casting!

# Creating Your Own Generics

// from java.util code

// Note that E can now stand for any class

```
public interface List<E> {
 void add(E x);
 Iterator<E> iterator();
}
```

```
public interface Iterator<E> {
 E next();
 boolean hasNext();
}
```

# Specific Generics :D

- We can get even more specific when declaring Generics and say that they can only be subclasses of a certain type, e.g.
- `class DogList<E extends Dog>`
- `class Arithmetic<E extends Number>`

# DRY Generics

DRY - “Don’t Repeat Yourself” - a fundamental tenet of writing good code

However, look back at WideningDemo.java - SO MUCH REPETITION.

Generics can help us DRY up code - reducing the amount of copy/pasted code we need to do. Numbers aren't actually a good example for various reason - lots of special things to consider.