Intro To Java

WELCOME TO ELEVEN FIFTY

Syllabus

Section 1: Hello World

- Syntax & Operators
- Primitive Types
- Conditionals

Section 2: Objects

- Arrays
- Objects
- Interfaces
- Abstract Classes

Section 3: Exploring Java

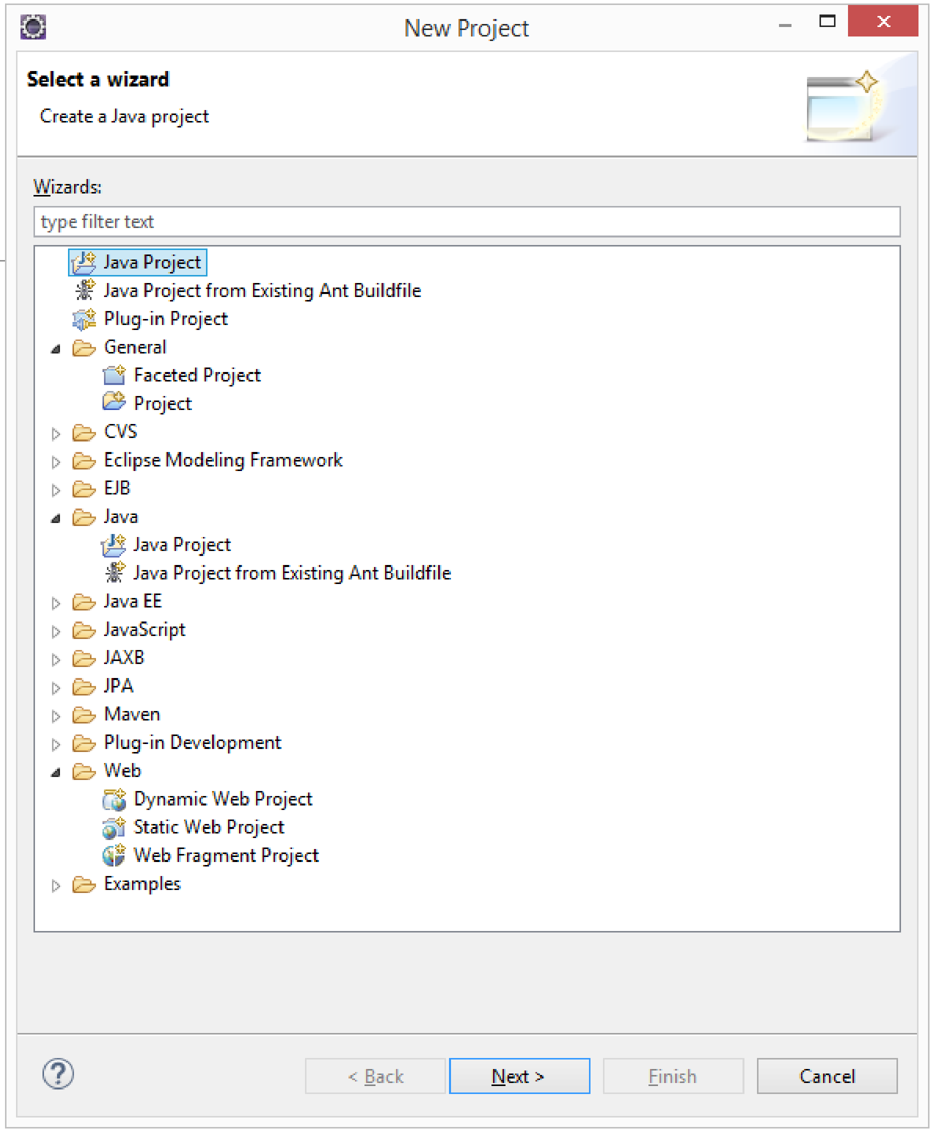
- Collections
- Generics
- Equality
- Dates
- Exceptions
- File I/O

What is Java

- Created by Sun microsystems in 1991, released 1995
- Came from C/C++
  - Syntax is similar
- Memory Management
- Runs on Billions of devices around the world
- One of the most popular programming languages in the world

Create a New Project

1. File -> New -> Project…
2. Select the project type you like
3. Name it ElevenFiftyIntro
4. Review Settings


New Project

Select a wizard

Create a Java project

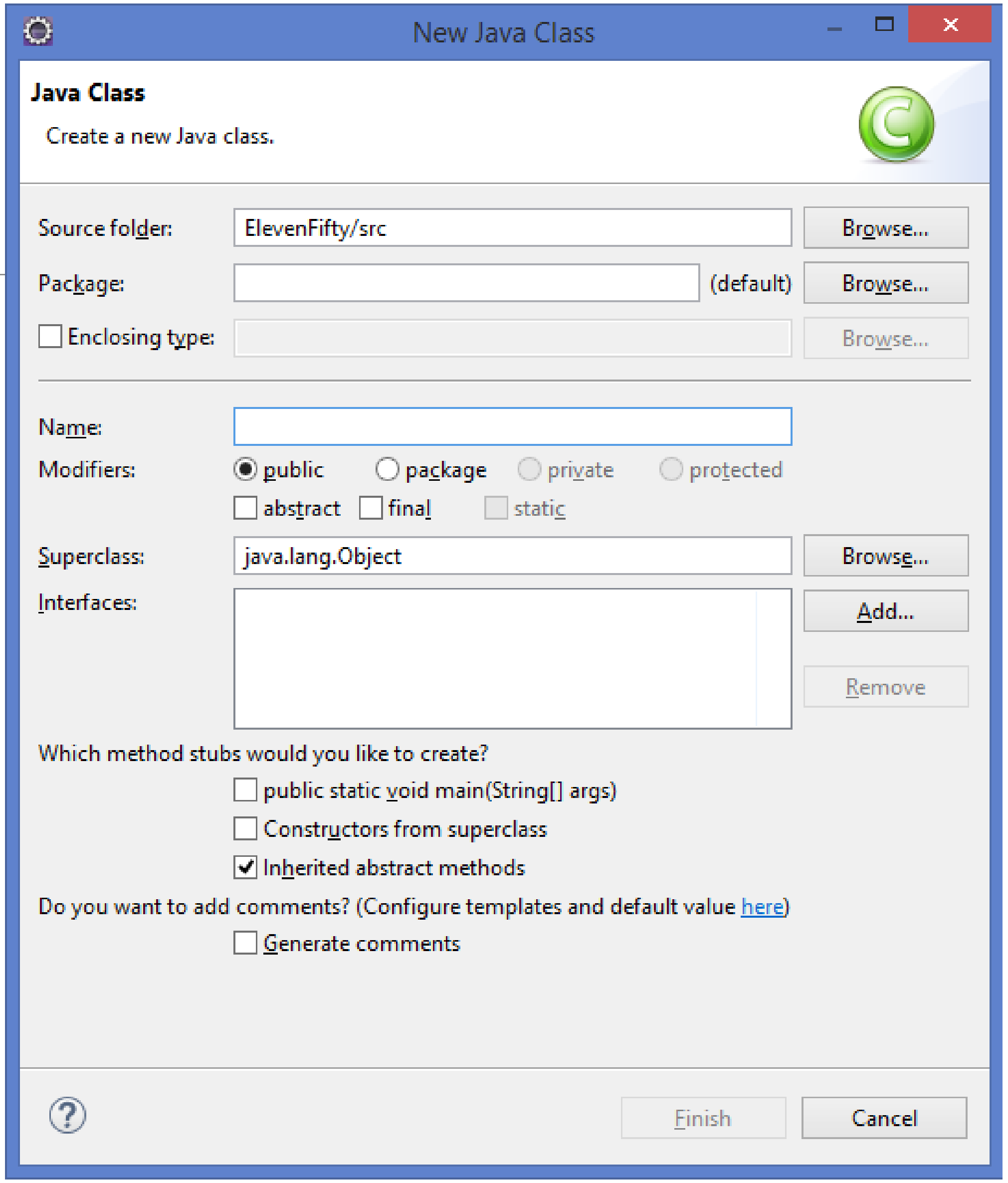
Wizards:

type filter text

- Java Project
- Java Project from Existing Ant Buildfile
- Plug-in Project
- General
  - Faceted Project
  - Project
- CVS
- Eclipse Modeling Framework
- EJB
- Java
  - Java Project
  - Java Project from Existing Ant Buildfile
- Java EE
- JavaScript
- JAXB
- JPA
- Maven
- Plug-in Development
- Web
  - Dynamic Web Project
  - Static Web Project
  - Web Fragment Project
- Examples

Create a New Class File

1. Right click on our new project
2. Highlight "New"
3. From the menu select "Class"
4. Enter a name for your class file
5. Check the box next to "public static void…"
6. Hit finish



New Java Class

Java Class

Create a new Java class.

Source folder: ElevenFifty/src [Browse...]

Package: (default) [Browse...]

☐ Enclosing type: [Browse...]

Name:

Modifiers: ● public ○ package ○ private ○ protected

☐ abstract ☐ final ☐ static

Superclass: java.lang.Object [Browse...]

Interfaces: [Add...] [Remove]

Which method stubs would you like to create?

☐ public static void main(String[] args)

☐ Constructors from superclass

☑ Inherited abstract methods

Do you want to add comments? (Configure templates and default value here)

☐ Generate comments

[Finish] [Cancel]

Saying Hello

1. Type System.out.println(“Hello world”); into our main method stub
2. Right click within our class file
3. Select the “Run” option
4. Run as Java Application
5. Look in the console

```
public static void main(String[] args){
    System.out.println(“Hello World”);
}
```

Basic Syntax

- Lines end in semicolons;
- Blocks may be used to define classes, interfaces, methods, conditionals, and more!
- Periods are used to call methods on objects or classes
- What happens in a code block, stays in the code block

```
// Denotes a line comment
//Start Block
{
  String variableName = “stuff”;
  variableName.methodName(parameters…)
}
// End block
```

Variable Syntax

access (keywords) Type name

```
public String variableName;
```

```
private final int anotherVariable;
```

Declaring Variables

- Starts with a variable type and name
- Use camelCase when naming
- Eclipse will help you avoid invalid names
- Examples

```
boolean trueFalse = false;
int intNumber = 3;
double floatingNumber = 3.141d;
String someText = “Enter text here”;
```

Primitive types

- boolean (true/false bit)
- byte (Integer)
- short (Integer)
- int (Integer)
- long (Integer)

- float (Floating point)
- double (Floating point)

- char (a character)
- String (a series of characters

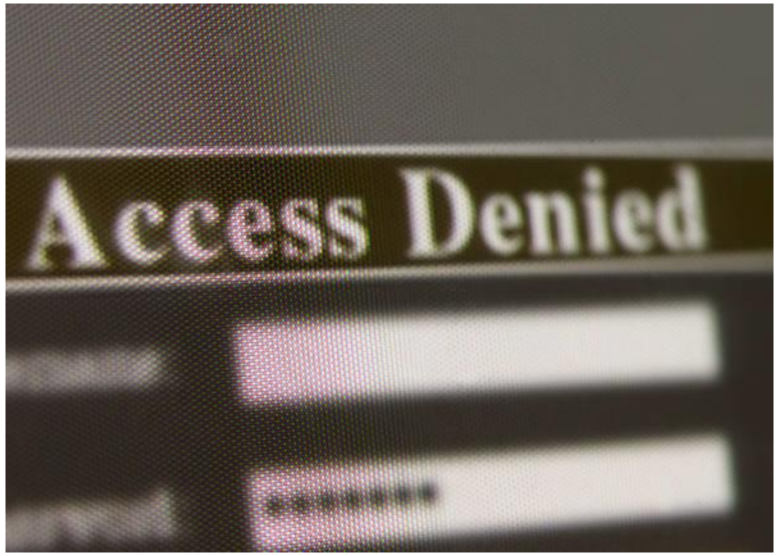
Access Modifiers

public
- All other classes may have access

protected
- Subclasses only

default
- Only classes within the current package have access
- Not used very often
- Not using another access modifier will result in default

private
- No other classes have access



Strings

- They can be any length of characters
- Use double quotes (“”) to create a string value
- Strings are objects with methods
  - use variableName.methodName() to access
- What is the output of the following lines?
  - String newString = “I AM LEARNING JAVA”;
  - System.out.println(newString.toLowerCase());
  - System.out.println(newString);
- String are also special for another reason, they are immutable!

Muta-huh?

mu·ta·ble
/ˈmyo͞otəbəl/
Adjective

liable to change.
“primitive variables in Java are mutable“

Purgatory related questions:

1. What happens to an old String?
2. What about an Object?

In order to “save” our string as a lower case:

```
newString = newString.toLowerCase());
```

Operating on Variables

- Assignment operator =
- Equal ==
- Not Equal !=
- Logical And &&
- Logical Or ||
- Math * / % + -
- Assignment + Math += -= /= *= %=

1. boolean shouldBeTrue = true;
2. boolean logicalFalse = shouldBeTrue && false;
3. boolean logicalTrue = shouldBeTrue || false;
4. double average = 90/100 * 100;
5. int remainder = 20 % 3;

Conditionals (If)

HUMAN

My son and I would like to go to the park tomorrow, but the weather looks like it could rain. If the weather is nice we will go to the park. Otherwise we will go to the Children's Museum of Indianapolis.

JAVA

```
String destination = “”;

if(weather == “good”){
     destination = “park”;
} else {
     destination = “museum”;
}
```

- Conditionals will evaluate Boolean expressions to determine what to do next.
- You may also add multiple conditions to an if-block by using “else if” and another condition.
- Once the first condition in the block is met, no others will be evaluated.
- “Else” conditions are not necessary.

Variable Scope

1. Static (Class)
   - Declared statically and will be associated to the class and not any implementation
2. Member (Instance)
   - Each class implementation has it’s own copy
3. Parameter
   - Live for the life of the method
4. Local
   - Declared within blocks and last as long as the block

Hint:
Code blocks can be found by looking for curly brackets { }

```java
public class Variables {
    private static final String CLASS_SCOPE = "CLASS";

    private String instanceScope = "INSTANCE";

    public String method(String parameterScope){
        String localScope = "LOCAL";

        if(parameterScope.equalsIgnoreCase(localScope)){
            String localScope2 = "LOCAL";
        } else {
            String localScope2 = "LOCAL";
        } //localScope2 no longer exists
    }
}
```

Loops

1. for(initialization expression ; termination expression ; increment expression) { … }
2. while (expression) { … }
3. do { … } while(expression)
4. for(variable : <Iterable>) { … }
   - Referred to as a “for each” loop
   - Works with arrays and collections

Try to avoid nesting loops

Calculating 7!

WHILE LOOP

```
int counter = 7;
int total = 1;
while(counter > 1){
     total *= counter;
     counter--;
}
```

FOR LOOP

```
int total = 1;
for(int i = 7; i > 1; i--){
     total *= i;
}
```

1. Both of these loops count down. What would get modified to make them count up?

2. How can we change these code snippets to take a variable instead of just a hardcoded 7?

Creating Methods

- We’ve already created a main method
- Methods are declared with the following
  1. Access modifier (public)
  2. Method level modifiers (static, final)
     - More on these later
  3. Return type (void or type)
  4. Name (main)
  5. Parameter List

public static void main(String[] args) { … }

Lets take our factorial code and create a method we can call from our main method

- Hint: make sure the method is static.

$$\begin{aligned}
0! &= 1\\
1! &= 1\\
2! &= 1 \cdot 2 = 2\\
3! &= 1 \cdot 2 \cdot 3 = 6\\
4! &= 1 \cdot 2 \cdot 3 \cdot 4 = 24\\
5! &= 1 \cdot 2 \cdot 3 \cdot 4 \cdot 5 = 120\\
6! &= 1 \cdot 2 \cdot 3 \cdot 4 \cdot 5 \cdot 6 = 720
\end{aligned}$$

Returning Data

- Method declarations require a type to be returned
- The keyword “return” will return an object (or nothing) from a code block for a method
- If you want to return nothing use “void”
- Only one return object

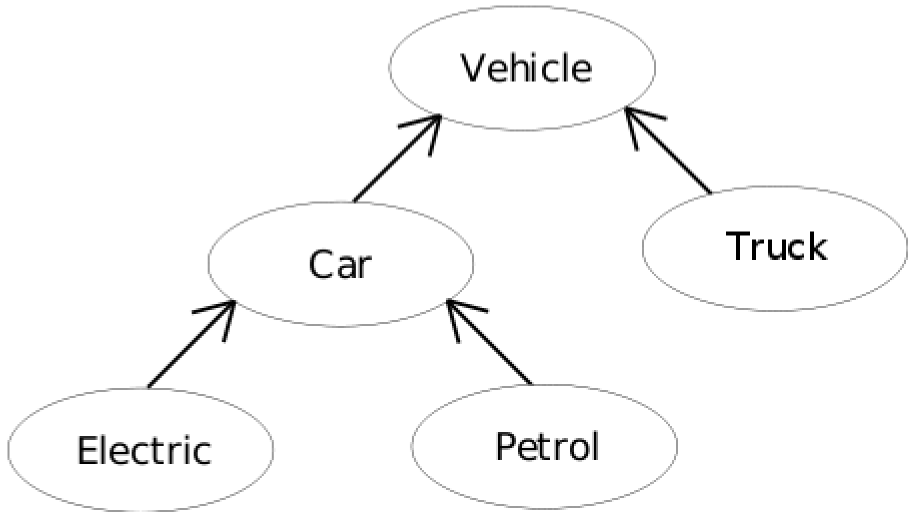
More Practice: Euler Problems

Using what we've just learned, create methods that solve the following problems:

1. Print the sum of all the multiples of 3 or 5 below 1000.
2. Print out the first 50 numbers of the Fibonacci sequence.

Inheritance

- Objects can access elements of the parent objects
- Access only goes upward
- Access modifiers and keywords can limit child access
- Applies to both extensions and implementations (more later)



Encapsulation

- Objects should provide access to only the things the caller should need access to
- Need to know basis
- Controlled with access modifiers
- What should a “client” of my object need access to?

Polymorphism

- An object may be assigned a value of a child class or implementation
- Only methods and variables associated to the object type may be access
- Child object may provide their own implementation of called methods
- Allows us to do some neat things with collections (more later)

"Now Speak!"
"Woof"
"Quack"
"Meow"

Cohesion and Coupling

COHESION

- Does the object represent one central purpose?
- High Cohesion refers to one object that accomplishes one goal (Good!)
- Higher Cohesion makes the object easy to understand and maintain

COUPLING

- Does the object heavily rely on other objects to perform its function?
- Loose Coupling is usually preferred over tight coupling (Good in most cases)
- Loose Coupling may also lead to more code reuse

Primitive Types as Objects

- Each of the primitive types has an object partner
- Autoboxing takes a primitive and automagically converts it to that types object and vice-versa
- Null objects do not have a primitive value

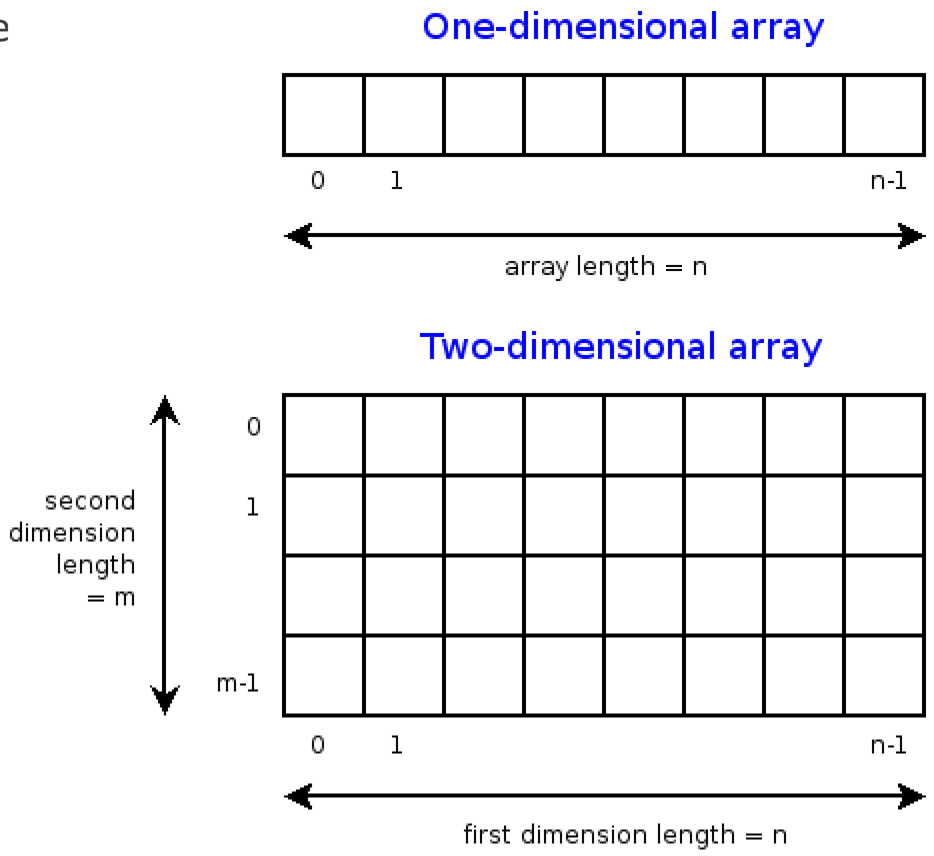
```
Integer objectInt = 7;

Double objectDouble = 3.141d;
```

Primitive Arrays

- Arrays are objects that can hold a predefined number of value
- Double square brackets ([]) tell Java the object is an array
- Upon instantiation, a length is required between the brackets
- Array indexes start at 0

What is the output of the code snippet?

```
int[] array;
int n = 42;
array = new int[n];
for(int i = 0; i < array.length; i++){
     array[i] = i;
}
System.out.println(array[0]);
System.out.println(array[42]);
```



null

Instantiation & Constructors

- Declaration is the same.
- Creating objects uses the **new** keyword
- Java uses the object's constructor to instantiate the object
- Objects may have multiple constructors to use
- Constructor arguments passed within parenthesis

Examples:

- Object newObject = new Object();
- Date myDate = new Date(System.currentTimeInMillis);
- File myFile = new File("file location");

Constructors

- Special methods that are run by the “new” keyword
- Sets up objects
- Empty constructor is implied unless you create more constructors
- Constructors must have unique parameter sets

A Note on Garbage Collection

- Handles the memory management for Java applications
- Removes objects and variables in application memory that are no longer being used
- Best just to leave it be

Fruit Classes

```
public class Orange {
   private String color = "orange";
   private String name = "Orange";
   private String origin = "Florida";
   private int weight = 200; // grams
```

```
public class Apple {
   private String color = “red";
   private String name = “Apple";
   private String origin = “Michigan";
   private int weight = 150; // grams
```

- It is rare to have public member variables
- Methods used to get and set object variables as getters and setters
- The coding standard is to use “get” or “set” followed by the name of the variable
- Keep any logic out of these methods
- If you don’t want the variable accessed, omit the getter or setting

```
public String getName() { return name; }

public void setName(String name) { this.name = name; }
```

Create all the Fruit!

1. Create a class to represent your favorite fruits.
2. Create fields for name, classification, color, origin, and weight.
3. Can you think of more fields?

Extends

- Our new fruit objects are extending the Object class
- There are times where we may want to extend another class instead
- The extends keyword used after the name of the class will allow you to create a subclass
- You can only extend one class at a time

```
public class Orange extends Fruit { … }
```

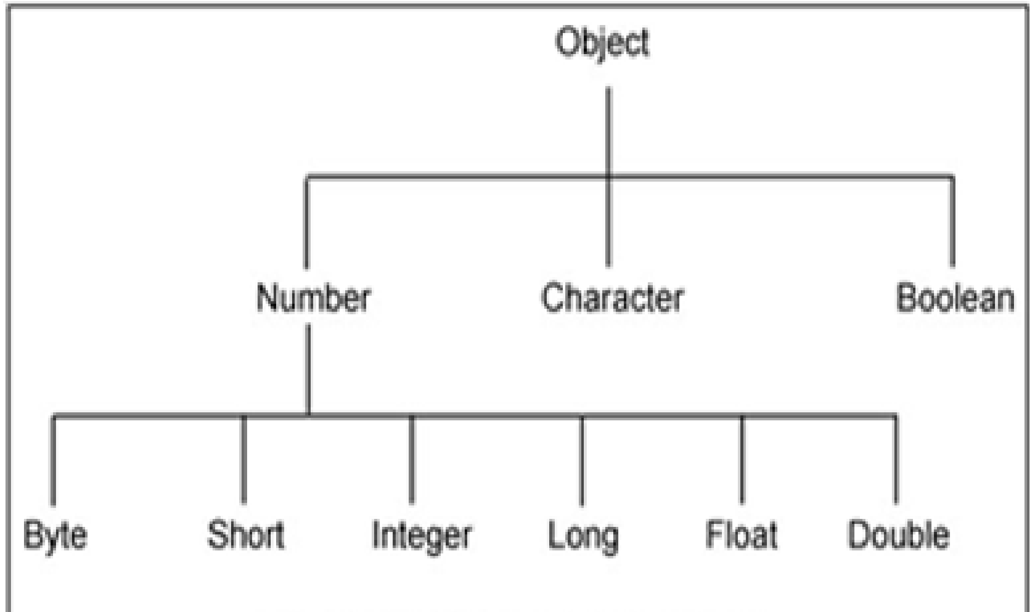


Figure : Wrapper classes Hierarchy

Override all the things!

- Parent methods may be overridden
- Protected and public methods only
- The keyword **this** will always refer to the object you are working in
- The keyword **super** will refer to a parent class
- Constructors can also be overridden

toString

- public String toString() { ... }
- Commonly overridden method from Object
- You supply the implementation
- I prefer JSON, but that is a tale for another day.

```
{
   "employees": [

      {
         "firstName": "John",
         "lastName": "Doe"
      },

     {
         "firstName": "Anna",
         "lastName": "Smith"
      },

      {
         "firstName": "Peter",
         "lastName": "Jones"
      }

   ]
}
```

It’s The Final Keyword…

Variable

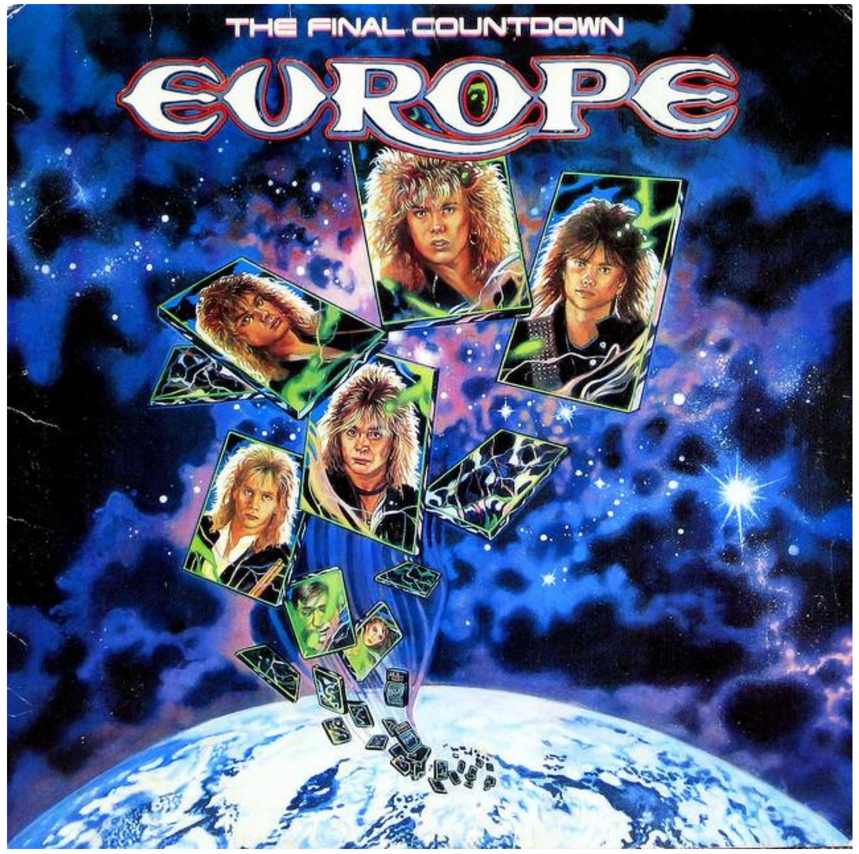
- Prevents the value from being changed.
- When combined with static it can create a safe constant value

Method

- Final methods cannot be overridden

Class

- Final classes may not be extended
- See Math.java



Abstract Classes

- Allows for a skeletal version of common functionality that can be shared amongst child implementations
- Classes can be declared **abstract**
  - public abstract class BaseFruit { … }
- Abstract classes cannot be instantiated
- Abstract classes may declare methods as abstract as well
  - protected abstract doSometing();

Create Base Fruit

1. Create a new class BaseFruit
2. Check the abstract box in the dialog box
3. Copy the redundant properties into BaseFruit
    1. Be sure they are now protected
4. Within each fruit object, extend BaseFruit
5. Remove properties that exist in BaseFruit

```java
public abstract class BaseFruit {
   protected String color = "";
   protected String name = "none";

   //Getters and setters not shown


public class Orange extends BaseFruit {

  //This is now the entire class file.  Neat huh?
  public Orange() {
    this.name = "Orange";
    this.color = "orange";
  }
```

Interfaces

- Interfaces allow developers to define method signatures that will need to be implemented by class files
- The signatures will outline functionality by specifying a return type, method, name, and parameters.
- All interface methods are public
- Classes may implement multiple interfaces
- Keep your interfaces clean and to the point

What interfaces could we create to help further define the fruit?

Create Peelable and Pitable Interfaces

1. Right click as if to create a class file
2. Select Interface
3. Give the interface a name and click finish
4. Create method stubs
   - void peel() for Peelable
   - void pit() for Pitable
5. Use the **implements** keyword to have a class declare it implements the interface
   - public class Orange extends BaseFruit implements Peelable { … }

Abstract vs Interface

INTERFACE

- Cannot be instantiated
- Classes can implement many interfaces
- Defines methods to be implemented
- No member variables
- No method implementations
- Can extend other interfaces, but cannot change the parent's defined methods

ABSTRACT CLASS

- Cannot be instantiated directly
- Classes may only extend one class at a time
- Can define methods as well as provide common implementations
- May contain member variables
- May contain implemented methods
- Can override parent object methods

Collections

- Implementations of various data structures
- Dynamically built and sized
- Plenty of helper methods and classes
  - Collections.java for example

Commonly used collections:

1. Set
   - A collection without duplicates
   - Common Implementation is HashSet
2. List
   - Ordered collection
   - Common Implementation is ArrayList
3. Map
   - Unique keys that are mapped to values
   - Common Implementation is HashMap

There are more types. Find them in the Javadoc

Generics

- The compiler to help you avoid errors
- After compilation the information is lost
- Allows implementations to make some assumptions if it chooses
  - Collections don't care what they store

- So instead of :

```
List fruitList = new ArrayList();
```

- We can

```
List<Fruit> fruitList = new ArrayList<Fruit>();
```

LUNCHEON LOAF
LUNCHEON
LOAF
INGREDIENTS: PORK, CHICKEN, SALT, FLAVORINGS, SUGAR, WATER, SODIUM, PHOSPHATE, SODIUM NITRITE.

== is not equal to equals()

```
Orange o1 = new Orange();

Orange o2 = new Orange();

//What is the output of the following?

System.out.println(o1 == o2);
```

Equals & HashCode

- Provides a method to customize object equality
- Review the Javadoc for each method
- Some objects require proper implementations
  - Hash collections for example
- Equals Implementations should be
  - **Reflexive** x = x
  - **Symmetric** x = y then y = x
  - **Transitive** x = y and y = z then x = z
  - **Consistent** 100% of calls are the same
- Hashcode
  - Equal objects should have equal hash codes

TODO: How should our Fruit Classes handle this situation?

Sorting

- Collections.sort()
  - Comparable interface
  - Comparator interface
- Picks an efficient sort algorithm based on collection size
- Just provide an implementation that can compare objects
  - Negative if object 1 < object 2
  - 0 if object1 == object 2
  - Positive if object 1 > object 2

TODO: Lets make our fruit comparable.

How to Read an Exception

1. Read the Exception class and the message

2. Is there a “Caused By”

3. Find your code (It’s most definitely you)

Exceptions

```
java.lang.RuntimeException: ManagedObjectCache(binaryContentCache) failed PUT(key, value) at
    com.chacha.cache.ManagedObjectCache.put(ManagedObjectCache.java:115) at
    com.chacha.cache.CacheRefresher.runLoop(CacheRefresher.java:59) at
    com.chacha.thread.ManagedThread.run(ManagedThread.java:130) at
    java.util.concurrent.ThreadPoolExecutor$Worker.runTask(ThreadPoolExecutor.java:886) at
    java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:908) at java.lang.Thread.run(Thread.java:662) Caused by:
    java.lang.RuntimeException: ManagedObjectCache(managedMemcachedCache) failed PUT(key, value) at
    com.chacha.cache.ManagedObjectCache.put(ManagedObjectCache.java:115) at
    com.chacha.cache.ManagedObjectCache.put(ManagedObjectCache.java:96) ... 5 more Caused by: java.lang.RuntimeException: Failure
    setting memcached key/value at com.chacha.cache.MemcachedCache.put(MemcachedCache.java:116) at
    com.chacha.cache.MemcachedCache.put(MemcachedCache.java:127) at com.chacha.cache.CriteriaCache.put(CriteriaCache.java:29) at
    com.chacha.cache.ManagedObjectCache.put(ManagedObjectCache.java:96) ... 6 more

    Caused by: java.lang.IllegalArgumentException: Cannot cache data larger than 1MB (you tried to cache a 2071771 byte object) at
    net.rubyeye.xmemcached.transcoders.CachedData.<init>(CachedData.java:95) at
    net.rubyeye.xmemcached.transcoders.SerializingTranscoder.encode(SerializingTranscoder.java:229) at
    com.chacha.cache.memcached.PerformanceTrackingTranscoder.encode(PerformanceTrackingTranscoder.java:33) at
    net.rubyeye.xmemcached.command.text.TextStoreCommand.encodeValue(TextStoreCommand.java:197) at
    net.rubyeye.xmemcached.command.text.TextStoreCommand.encode(TextStoreCommand.java:153) at
    net.rubyeye.xmemcached.impl.MemcachedTCPSession.wrapMessage(MemcachedTCPSession.java:173) at
    com.google.code.yanf4j.core.impl.AbstractSession.write(AbstractSession.java:381) at
    net.rubyeye.xmemcached.impl.MemcachedConnector.send(MemcachedConnector.java:498) at
    net.rubyeye.xmemcached.XMemcachedClient.sendCommand(XMemcachedClient.java:265) at
    net.rubyeye.xmemcached.XMemcachedClient.sendStoreCommand(XMemcachedClient.java:2546) at
    net.rubyeye.xmemcached.XMemcachedClient.set(XMemcachedClient.java:1312) at
    net.rubyeye.xmemcached.XMemcachedClient.set(XMemcachedClient.java:1370) at
    com.chacha.cache.MemcachedCache.put(MemcachedCache.java:105) ... 9 more
```

Common Exceptions

- NullPointerException
  - Trying to access an object that is null
- ArrayIndexOutOfBoundsException
  - That index doesn’t exist!
- ClassCastException
  - Something is trying to turn an object into a class that is not valid
- StackOverFlowException
  - You probably created something that is infinitely operating. This is bad.
- RuntimeException
  - Wrapper for exceptions that happen during Runtime
- IOException
  - Something with I/O went badly

Checked vs Unchecked

- Unchecked exceptions usually happen during run time

- Checked exceptions force the calling method to handle

- Bottom Line
  - If a client can reasonably be expected to recover from an exception, make it a checked exception.

  - If a client cannot do anything to recover from the exception, make it an unchecked exception.

Handling Exceptions

- try { … } catch(Exception e) { … } finally { … }
- Methods can **throw(s)** exceptions
- You can create your own
- Should extend Exception and implement Throwable
- Finally and catch are optional, but very useful

```
try{
    //...code…
} catch(IOException ioe) {
    //...handling code…
} catch(Exception e) {
    //...handling code…
} finally {
    //…wrap things up…
}

public String doSomethingImportant() throws Exception { }

...
throw new IllegalArgumentException(“Bad value for foo”);
...
```

The Dark Side

1. Catch the most granular Exception you can
2. Make sure to log/handle exceptions somewhere
3. Avoid “swallowing” exceptions
4. Use finally blocks finalize items when necessary

Debug Mode

- Lets you watch while the application is running
- Line by line operation
- Debug Perspective
  - Stack
  - Variables
  - Breakpoints

Smoothies

Design and write the code for a smoothie machine using our ingredients.

- Make sure that it can take any amount of the fruits you gave it.
- Calculate the cost of the smoothie
- Bonus: Print instructions for making dynamically

Beyond Oracle

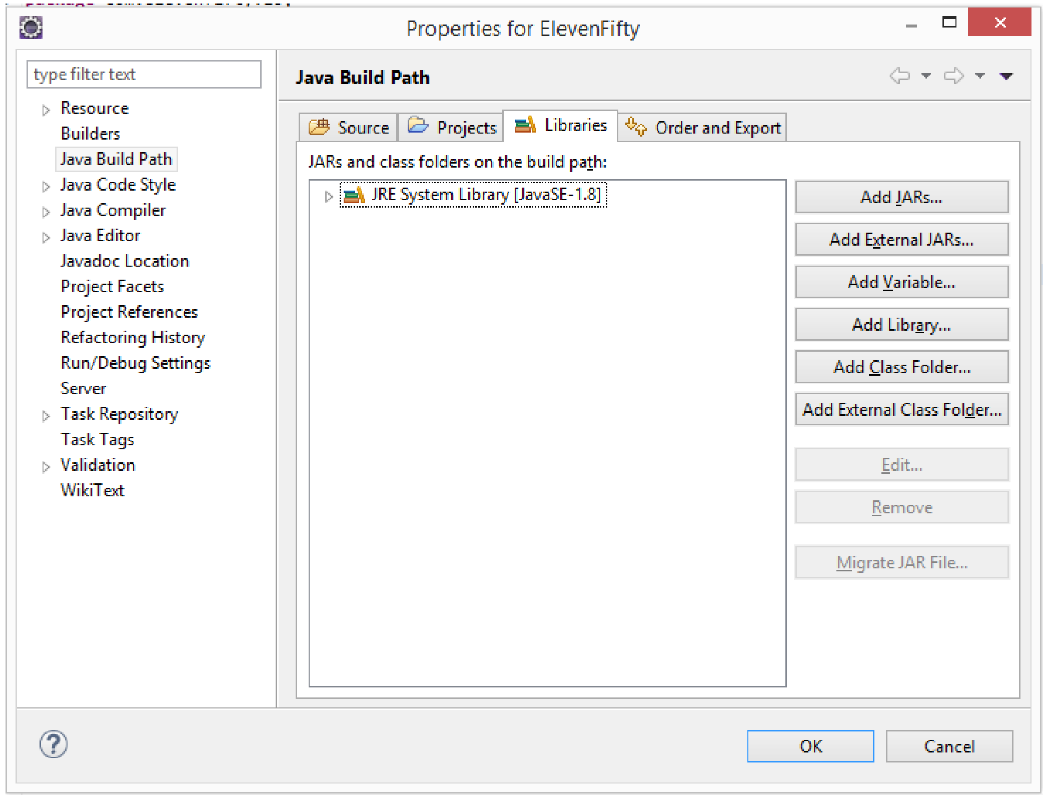
- Lots of 3rd party libraries
- Most (if not all) reputable libraries have Javadoc available
- There are even libraries to help applications manage libraries within a project
- Chances are someone has created a library to do what you would like to do
- Some libraries are open source

List of Common 3rd party Libraries

1. Apache Commons
   - Tons of great libraries to help with various tasks
2. Guava
   - Made by google
3. Jackson/GSON
   - JSON processing
4. Log4j and SLF4J
   - Better ways of handling logging
5. Spring Framework
   - Makes a lot of things easier. More on this soon

Including Libraries in your projects

1. Create a folder name “lib” in the root of the project
2. For now, move 3rd party jar files into the folder
3. Configure the build path for the project in Eclipse
4. Select “Libraries” and click on “Add Jar”
5. Select the Jar files you wish to add
6. Hit “ok”
7. Rebuild project
8. Profit



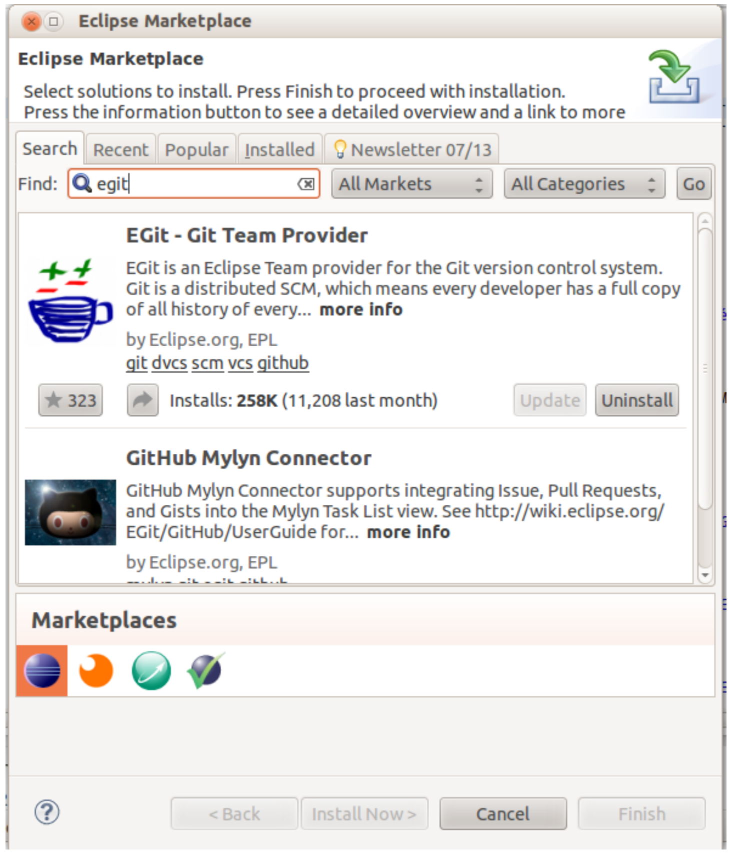
Making things easier

- Finding and adding all of those libraries and their dependencies can be quite taxing.
- Apache Ivy is an agile dependency manager
- Just tell Ivy what libraries you will require and it will take care of the rest
- Can be setup to allow corporate teams to share internal libraries

Apache IvyDE

To make things a little simpler, Apache has supplied a plugin for Eclipse that will help manage Apache Ivy. Lets take the time to explore the Eclipse marketplace.

1. Help -> Eclipse Marketplace
2. Search for “Apache IvyDE”
3. Install and restart eclipse



Eclipse Marketplace

Select solutions to install. Press Finish to proceed with installation.
Press the information button to see a detailed overview and a link to more

Find: egit

EGit - Git Team Provider

EGit is an Eclipse Team provider for the Git version control system. Git is a distributed SCM, which means every developer has a full copy of all history of every... more info

by Eclipse.org, EPL

git dvcs scm vcs github

Installs: 258K (11,208 last month)

GitHub Mylyn Connector

GitHub Mylyn Connector supports integrating Issue, Pull Requests, and Gists into the Mylyn Task List view. See http://wiki.eclipse.org/EGit/GitHub/UserGuide for... more info

by Eclipse.org, EPL

Using IvyDE

1. Configure Apache Ivy For Project
   - Right click Project and “Configure”
     - Add IVY Dependency Management
   - Create ivy.xml file
   - Right click ivy.xml and “Add Ivy Library…”
     - Select project and click on your ivy file
     - Hit Ok
   - IVY menu for project should have lots of options now
2. No you can do the following
   - Add dependencies
   - Resolve
   - Clear caches
   - Exclude

1. Add the following to your Ivy.xml

```
<dependency org="org.springframework"
name="spring-core" rev="4.1.0.RELEASE"/>
```

2. Under the Project’s Ivy menu select Resolve
3. Explore your new libraries

HINT: Find this text at http://mvnrepository.com/

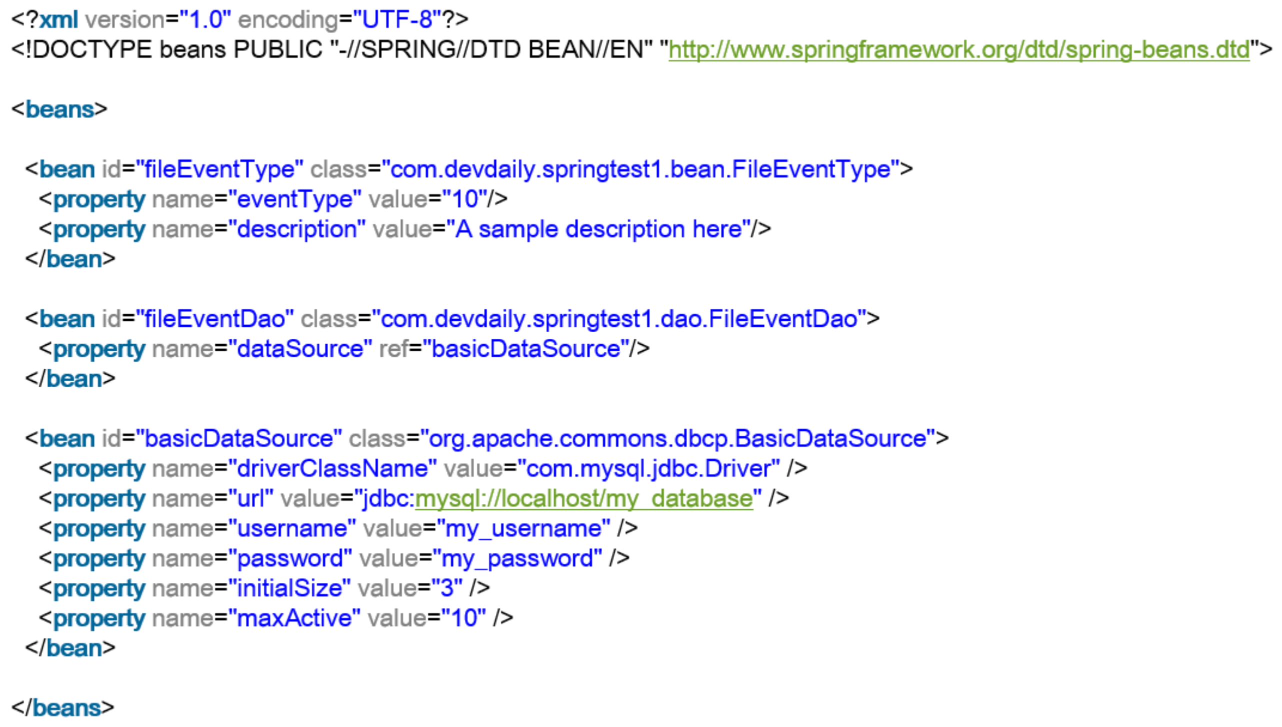
Using Spring

- Lets create a Main.java class with a main method
- The code on right will be able to load the spring configuration
- The context will be your access to Spring’s BeanFactory

```
ApplicationContext context = new
ClassPathXmlApplicationContext(new String[]
{“applicationContext.xml"});
```

Spring Configuration

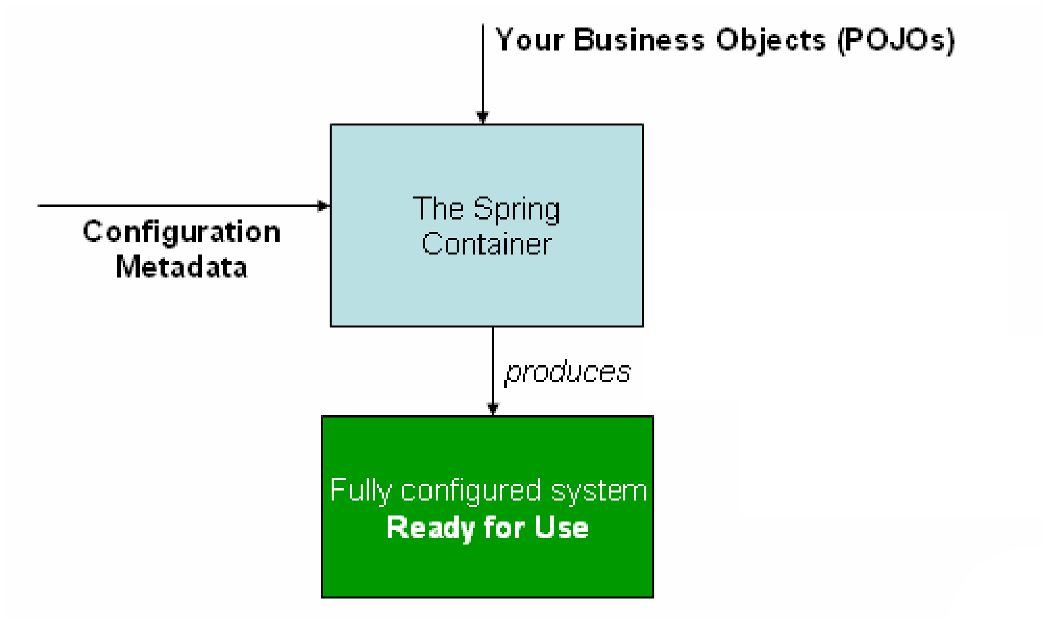
- Usually done with XML
- Can import files
- Allows for abstract configurations
  - Parent attribute specifies a beans parent
- Scope attribute
  - Singleton only one instance per application
  - Prototypes will create a new instance every time
- Give each bean a unique id



```xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN//EN" "http://www.springframework.org/dtd/spring-beans.dtd">

<beans>

  <bean id="fileEventType" class="com.devdaily.springtest1.bean.FileEventType">
    <property name="eventType" value="10"/>
    <property name="description" value="A sample description here"/>
  </bean>

  <bean id="fileEventDao" class="com.devdaily.springtest1.dao.FileEventDao">
    <property name="dataSource" ref="basicDataSource"/>
  </bean>

  <bean id="basicDataSource" class="org.apache.commons.dbcp.BasicDataSource">
    <property name="driverClassName" value="com.mysql.jdbc.Driver" />
    <property name="url" value="jdbc:mysql://localhost/my_database" />
    <property name="username" value="my_username" />
    <property name="password" value="my_password" />
    <property name="initialSize" value="3" />
    <property name="maxActive" value="10" />
  </bean>

</beans>
```

Inversion of Control

- IoC is also known as dependency injection
  1. Beans define their dependencies
  2. The container then *injects* those dependencies when it creates the bean
- This process is fundamentally the inverse of the bean itself controlling the instantiation or location of its dependencies
- Saves you a lot of time instantiating and configuring objects and their dependencies



Factory Pattern

- Factories are objects that create objects for you
- Used when creation of objects may involve some internal (private) logic
- Helps avoid code coupling
  - heavy dependence on other code
- See Java's Calendar as an example

Smoothie Machine 2.0

1. Lets add our fruit beans to our new spring configuration
2. Now we can retrieve our fruit from Spring
3. We should probably add the smoothie machine itself to the application context