Annotations & Reflection

CSE219, Computer Science III

Stony Brook University

http://www.cs.stonybrook.edu/~cse219

Annotations and Reflection?

- Features of some languages

- Programmer conveniences

- Useful in checking inheritance

- Alternate development modes

Annotations

- Remember @Override

- Remember JUnit?
  - @Before, @After, @Test
- @ is Java's notation for the start of an annotation
  - like @author for javadoc
- What are they?
  - metadata
  - provide data about a program

What are annotations used for?

- Information for the compiler
  - detect errors
  - suppress warnings.
- Compile-time and deployment-time processing
  - for IDEs and other tools
  - generate code, XML files, etc.
- Runtime processing
  - some annotations are used at runtime.

Annotations can have field names and data

```
@Author(
    name = "Benjamin Franklin",
    date = "3/27/2003"
)
class MyClass()



@SuppressWarnings(value = "unchecked")
void myMethod() { ... }
```

Where can annotations be used?

- Declarations of classes, fields, methods, etc.
- Java SE 8 also has type annotations:
  - Class instance creation expression:

    ```
    new @Interned MyObject();
    ```

  - Type cast:

    ```
    myString = (@NonNull String) str;
    ```

  - implements clause:

    ```
    class UnmodifiableList<T> implements
            @Readonly List<@Readonly T> { ... }
    ```

  - Thrown exception declaration:

    ```
    void monitorTemperature() throws
            @Critical TemperatureException { ... }
    ```

(c) Paul Fodor

Why do we care about annotations?

- Tools love to use them
  - JUnit
  - Javadoc
  - Web-related Tools:
    - Java Persistence API (JPA)
      - describes the management of relational data in applications
    - Application Servers

(c) Paul Fodor

Annotations Look-Up

- Scattered in the Java API. Examples:

http://docs.oracle.com/javaee/8/api/javax/annotation/package-summary.html

http://docs.oracle.com/javaee/8/api/javax/faces/bean/package-summary.html

- Via cheat sheets:

Java EE 7 Annotations

Alternatives for management

CDI	JSF	EJB

CDI: javax.inject

CMF	@Inject
	@Named(value="")
	@Singleton

CDI: javax.enterprise.context

TMF	@ApplicationScoped
TMF	@SessionScoped
TMF	@ConversationScoped
TMF	@RequestScoped
TMF	@Dependent

CDI: javax.enterprise.inject

TMF	@New(value=Class.class)
TMF	@Alternative
TMF	@Any
MF	@Produces, @Disposes

JSF management: javax.faces.bean

T	@ManagedBean(name="", eager=false)
T	@CustomScoped(value="")
T	@NoneScoped
T	@ApplicationScoped
T	@SessionScoped
T	@ViewScoped
T	@RequestScoped
F	@ManagedProperty(name="", value="")
T	@ReferencedBean(name="")

EJB injection: javax.ejb

TMF	@EJB(name="", beanInterface=Interface.class, mappedName="", lookup=""|| beanName="", description="")
T	@EJBs(@EJB[])

Resource injection: javax.annotation

TMF	@Resource(name="", type=Class.class, authenticationType= [AuthenticationType.CONTAINER, APPLICATION], shareable=true, lookup="", mappedName="")
T	@Resources(@Resource[])

EJB Types: javax.ejb

Session beans

T	@Stateless(name="ClassName")
T	@Stateful(name="ClassName")
T	@Singleton(name="ClassName")
T	@Local(Class.class[]) on EJB, @Local on interface
T	@Remote(Class.class[]) on EJB, @Remote on interface
T	@LocalBean
TM	@Asynchronous
TM	@Lock([LockType. WRITE,READ])
T	@ConcurrencyManagement([CONTAINER, BEAN])
T	@DependsOn(String[])
T	@Startup

Non-session beans

T	@MessageDriven(name="ClassName", activationConfig=@ActivationConfigProperty[])
	@ActivationConfigProperty(propertyName="", propertyValue="")
T	@ManagedBean(value="") [in javax.annotation.*]

Timeouts: javax.ejb

TM	@AccessTimeout(value="0",unit=MILLISECONDS)
T	@StatefulTimeout(value="0",unit=MINUTES)
M	@Timeout
M	@Schedule(year="*", month="*", bimonthly="*", dayOfWeek="*", hour="0", minute="0", info="", persistent=true, timezone="")
M	@Schedules(@Schedule[])

Transaction: javax.ejb

T	@TransactionManagement([CONTAINER, BEAN])
TM	@TransactionAttribute([TransactionAttributeType. MANDATORY, REQUIRED, REQUIRES_NEW, SUPPORTS, NOT_SUPPORTED, NEVER])
M	@AfterBegin
M	@BeforeCompletion
M	@AfterCompletion

Lifecycle: javax.ejb

M	@PostConstruct [in javax.annotation.*]
M	@PreDestroy [in javax.annotation.*]
M	@PostActivate
M	@PrePassivate
M	@Remove(retainIfException=false)

Interceptors: javax.interceptor

TM	@Interceptors(Class.class[])
TM	@ExcludeDefaultInterceptors
M	@ExcludeClassInterceptors
M	@AroundInvoke
M	@AroundTimeout
T	@Interceptor [only required with @InterceptorBinding]

Security: javax.annotation.security

T	@RunAs(String rolename)
T	@DeclareRoles(String[])
TM	@RolesAllowed(String[])
TM	@PermitAll
TM	@DenyAll

```
-- Possible source file layout  for web app -
1   lib/            [potentially copied to /lib/ inside an EAR]
    `-- extra.jar   [jar shared between all modules]
2   src/java/    [potentially packaged as EJB-JAR inside EAR -
    |-- ValidationMessages.properties          or under
    |-- JSFStrings.properties          WEB-INF/classes/
    |-- META-INF/                      inside WAR]
    |   |-- persistence.xml    [for JPA config]
    |   `-- ejb-jar.xml        [for deployment descriptors]
    `-- com/
        `-- myBusiness/
            |-- entities/
            |   `-- Entities.java
            `-- EJBs.java
    src/webapp/  [potentially packaged as a WAR inside an EAR]
3   |-- WEB-INF/
    |   |-- beans.xml          [for CDI config]
    |   |-- faces-config.xml   [for JSF config]
    |   `-- web.xml            [for Servlet 2.5 config]
    |-- resources/
    |   |-- css/
    |   |   `-- standard.css
    |   `-- javascript/
    |       `-- standard.js
    `-- jsfpages.xhtml
```

EAR class loader levels

Legend

TCMF Annotation for Type, Constructor, Method, Field
VALUE Default Value

Java EE 7 Annotations Cheat Sheet

Version 1.2 ©2005,2011 Philipp Meier
Version 1.5 (2013-06-27) by Chris Rennie, based on
Java EE 7 API Doc: EJB 3.2, JSF 2.2, JPA 2.1
www.physics.usyd.edu.au/~rennie/

8

Annotations

- Annotation Types Used by the Java Language
  - The predefined annotation types defined in java.lang are @Deprecated, @Override, and @SuppressWarnings.
    - @Deprecated annotation indicates that the marked element is deprecated and should no longer be used.
      - The compiler generates a warning whenever a program uses a method, class, or field with the @Deprecated annotation.

```
// Javadoc comment
/**
 * @deprecated
 * explanation of why it was deprecated
 */
@Deprecated
static void deprecatedMethod() { ... }
```

Annotations

- @Override annotation informs the compiler that the element is meant to override an element declared in a superclass.

```
// mark method as a superclass method
// that has been overridden
@Override
int overriddenMethod() { ... }
```

- @SuppressWarnings annotation tells the compiler to suppress specific warnings that it would otherwise generate.

```
// use a deprecated method and tell
// compiler not to generate a warning
@SuppressWarnings("deprecation")
void useDeprecatedMethod() {
    // deprecation warning
    // - suppressed
    objectOne.deprecatedMethod();
}
```

(c) Paul Fodor

Declaring an Annotation Type

- Define the annotation type:

```
@interface ClassPreamble {
    String author();
    String date();
    int currentRevision() default 1;
    String lastModified() default "N/A";
    String lastModifiedBy() default "N/A";
    // Note use of array
    String[] reviewers();
}
```

Declaring an Annotation Type

- After the annotation type is defined, you can use annotations of that type:

```
@ClassPreamble (
    author = "John Doe",
    date = "3/17/2002",
    currentRevision = 6,
    lastModified = "4/12/2004",
    lastModifiedBy = "Jane Doe",
    // Note array notation
    reviewers = {"Alice", "Bob", "Cindy"}
)
public class Generation3List extends List2{
    // class code goes here
}
```

(c) Paul Fodor

Declaring an Annotation Type

- To make the information in @ClassPreamble appear in Javadoc-generated documentation, when you define the annotation:

```
// import this to use @Documented
import java.lang.annotation.*;
@Documented
@interface ClassPreamble {

    // Annotation element definitions

}
```

Reflection

- A powerful programming feature
  - requires the ability to examine or modify the runtime behavior of applications running in the Java virtual machine.
- i.e. dynamically examine classes and objects
- Should be used only by developers who have a strong grasp of the fundamentals of the language.
- Can enable applications to perform operations which would otherwise be impossible.

Reflection

- Call methods at runtime that you didn't know existed at compile time.

- Isn't that polymorphism?
  - No, polymorphism uses inheritance and knows the overridden method signatures

- At runtime:

  - ask a Class what methods it has

  - call one of those methods

(c) Paul Fodor

Reflection Uses

- Extensibility Features
  - dynamically use classes not known at compile time
  - plug-ins, add-ons, etc.
  - complete flexibility
- Class Browsers and Visual Development Environments
  - i.e. display class properties
    - think the visual debugger
- Debuggers and Test Tools
  - Watch class values change

(c) Paul Fodor

Reflection

- It all starts with the **Class** class:
  - Every object in Java is a member of a class.
  - How do we get an object's **Class**?
  - **getClass()** method inherited from Object. Ex:

    Class c = "Hello".getClass();
  - Using **Class.forName** and a string. Ex:

    Class c2 = Class.forName("java.lang.String")
    - can throw **ClassNotFoundException**
  - Other methods:
    - **getSuperclass**
    - **getDeclaredClasses**
      - returns an array of Class object members declared by the class, but excludes inherited classes

        Class cls = Class.forName("ClassDemo");

        Class[] classes = cls.getDeclaredClasses();
    - **getEnclosingClass**
      - Returns the outer class of an inner class (or null if none)

17

The `Class` class has useful methods

Class Methods for Locating Fields

`Class` API	List of members?	Inherited members?	Private members?
getDeclaredField()	no	no	yes
getField()	no	yes	no
getDeclaredFields()	yes	no	yes
getFields()	yes	yes	no

Class Methods for Locating Methods

`Class` API	List of members?	Inherited members?	Private members?
getDeclaredMethod()	no	no	yes
getMethod()	no	yes	no
getDeclaredMethods()	yes	no	yes
getMethods()	yes	yes	no

Class Methods for Locating Constructors

`Class` API	List of members?	Inherited members?	Private members?
getDeclaredConstructor()	no	N/A[1]	yes
getConstructor()	no	N/A[1]	no
getDeclaredConstructors()	yes	N/A[1]	yes
getConstructors()	yes	N/A[1]	no

Fields

- Has a type and value

- Type is a **Class**

- Get/Set data via **get/set** methods

- Other useful classes
  - **Method**
  - **Constructor**

(c) Paul Fodor

Drawbacks of Reflection

- Performance Overhead
  - dynamic type resolution is expensive
  - certain Java virtual machine optimizations skipped
  - should be avoided in hot spots
- Security Restrictions
  - requires a runtime permission which may not be present when running under a security manager.
    - can't be used with **Applet**s
- Exposure of Internals
  - allows code to perform operations that would be illegal in non-reflective code
    - accessing private fields and methods
- can result in unexpected side-effects