Reflection

CSCI 4448/5448: Object-Oriented Analysis & Design

Lecture 37

Acknowledgement & Materials Copyright

- I’d like to start by acknowledging Dr. Ken Anderson
- Ken is a Professor and the Chair of the Department of Computer Science
- Ken taught OOAD on several occasions, and has graciously allowed me to use his copyrighted material for this instance of the class
- Although I will modify the materials to update and personalize this class, the original materials this class is based on are all copyrighted © Kenneth M. Anderson; the materials are used with his consent; and this use in no way challenges his copyright

Goals of the Lecture

- Definition of reflection and related terms
- Reflection in Java
- What can you do with it?
- Using it in Java and in general

Reflection?

- **Reflection** is the ability of a program to analyze or query itself and to modify itself during runtime
  - Querying is known as **introspection**
  - Changing state, modifying itself, or adding new behaviors is **intercession**
- Reflection can be either structural or behavioral
  - **Structural** refers to a program’s data and code
  - **Behavioral** refers to the runtime environment
  - For instance, behavioral intercession is modifying the runtime environment
- To query or modify execution state, the state must be represented in data elements – encoding execution state into data is called **reification**
- Programming languages are **reflective** if they implement any reflection capabilities

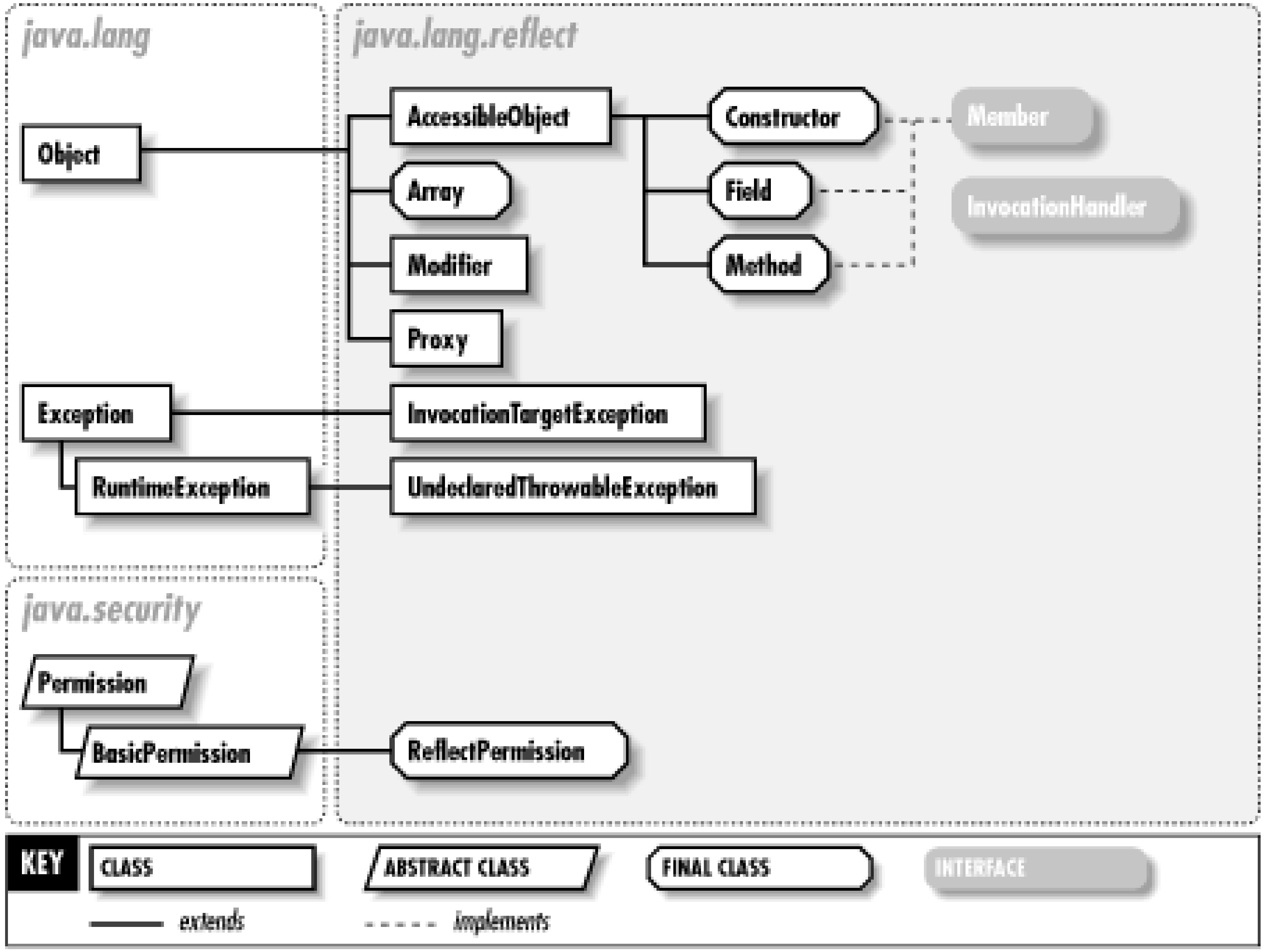
Reference [1]

Reflection in Java

- Java’s reflection is focused on introspection; intercession is limited
- Introspection features let you obtain class information about objects at runtime
    - Including fields, methods, modifiers, superclass, etc.
- Intercession in Java allows you to create an instance of a class whose name is not known until runtime
    - You can invoke methods on these instances and get or set attributes

- Java does NOT let you change the data structure of an object at runtime, you cannot add new fields or methods except at compile-time
- Behavioral intercession would be changing method execution or adding a method to a class at runtime, this is also NOT supported.

java.lang.reflect



Reference [2]

Things you can do with Java reflection

- If you have an object reference
  - you can determine the class name of the object
  - you can invoke its method knowing just the method's name and method's parameter types
- If you have a class name
  - you can know its full description, for example, its package name, its access modifiers, etc.
  - you can determine the methods defined in the class, their return type, access modifiers, parameters type, parameter names, etc.
  - you can determine all field descriptions of the class
  - you can determine all constructors defined in the class
  - you can create an object of the class using one of its constructors
- You can get or set the state of an object at runtime
- You can create an array of a type dynamically at runtime and manipulate its elements
- You can get the definition of a protected or final member and remove the protection!

Reference [1]

Class<T>

- Class<T> is a generic class, and a key to Java reflection
- It takes a type parameter
  - Class<String> is the class object for the String class
  - Class<?> is a class type whose class is unknown
- Class<T> lets you discover everything about a class at runtime
- When a object is created, Java loads the class byte code and creates an Class object to represent the byte code
- No matter how many objects of a class you create in a program, Java creates one Class object for each class loaded by a class loader in a JVM from a single module
  - In a JVM, a class is uniquely identified by its fully qualified name, class loader, and module
  - If two class loaders load the same class, the classes are considered to be different and objects from those classes will not be compatible

Reference [1]

Getting a reference to a Class object

- You can get a Class object reference by
  - Using a class literal
  - Using the getClass() method from the Object class
  - Using the forName() static method of the Class class

- A class literal is a class or interface name followed by ".class":

```
Class<Test> testClass = Test.class;
```

- An object reference will fail:

```
Test t = new Test();
Class<Test> testClass = t.class;    // compile-time error
```

- Class literals work for primitive types
  - int.class == Integer.Type   // same class object

Reference [1]

Getting a reference to a Class object

- Using the getClass() method from the Object class

```
Test testRef = new Test();
Class<Test> testClass = testRef.getClass();
```

- Using the forName() static method of the Class class
  - The static method forName() loads a class and returns the Class object reference
  - For a public class called Bulb in a named module

```
fullClassName = “modulename.Bulb”
Class<Bulb> c = Bulb.class;            // loads the class, does not initialize it
Class c = Class.forName(fullClassName);      // loaded and initialized
ClassLoader cLoad = MyTest.class.getClassLoader();
boolean init = false;
Class c = Class.forName(className, init, cLoad)    //loaded, not initialized (by flag)
Module m = MyTest.class.getModule();
Class c = Class.forName(m, className);     //loaded, not initialized
```

Reference [1]

Methods for getting class description

```
void examineClass(Class<?> cls) {
…
}
```

```
cls.isPrimative()
cls.isInterface()
cls.isAnnotation()
cls.isEnum()
cls.getSimpleName()
cls.getSuperclass()
cls.getInterfaces()
cls.getTypeParameters()
cls.getTypeName()
cls.getModifiers()
```

```
cls.getFields()     // accessible public fields only
cls.getDeclaredFields()
cls.getExceptionTypes()
cls.getMethods()
cls.getDeclaredMethods()
cls.getConstructors()
cls.getDeclaredConstructors()
```

Reference [1]

Methods for creating an object

- Create an object of a class using reflection
- No argument constructor for a class Person:

```
Class<Person> cls = Person.class;
Constructor<Person> noArgsCons = cls.getConstructor();
Person p = noArgsCons.newInstance();
```

- Get a specific constructor

```
Class<Person> personClass = Person.class;
// Get the constructor "Person(int, String)"
Constructor<Person> cons = personClass.getConstructor(int.class, String.class);
// Invoke the constructor with values for id and name
Person chris = cons.newInstance(1994, "Chris");
```

Reference [1]

Invoking methods on an object

```
//Given a class reference
Class<Person> personClass = Person.class;

// Create an object of Person class
Person p = personClass.newInstance();

// Get the reference of the setName() method
Method setName = personClass.getMethod("setName", String.class);

// Get a reference to all the class methods
Method[] methods = personClass.getMethods();

// Invoke the setName() method on p passing passing "Ann“ as a parameter
setName.invoke(p, "Ann");
```

Reference [1]

Finding Getters/Setters

- If we define getters and setters as:
  - A getter method has its name start with "get", takes 0 parameters, and returns a value
  - A setter method has its name start with "set", and takes 1 parameter
- Given that, here is code to find getter and setter methods from a class

Reference [3]

```
public static void printGettersSetters(Class aClass){
  Method[] methods = aClass.getMethods();

  for(Method method : methods){
    if(isGetter(method)) System.out.println("getter: " + method);
    if(isSetter(method)) System.out.println("setter: " + method);
  }
}

public static boolean isGetter(Method method){
  if(!method.getName().startsWith("get"))      return false;
  if(method.getParameterTypes().length != 0)   return false;
  if(void.class.equals(method.getReturnType()) return false;
  return true;
}

public static boolean isSetter(Method method){
  if(!method.getName().startsWith("set")) return false;
  if(method.getParameterTypes().length != 1) return false;
  return true;
}
```

When would you use Reflection?

- Instantiating arbitrary classes
  - For example, in a dependency injection framework, you probably declare that interface ThingDoer is implemented by the class NetworkThingDoer. The framework would then find the constructor of NetworkThingDoer and instantiate it.
- Marshalling and unmarshalling to some other format
  - For example, mapping an object with getters and settings that follow the bean convention to JSON and back again. The code doesn't actually know the names of the fields or the methods, it just examines the class.
- Wrapping a class in a layer of redirection or faking a class
  - Perhaps that List isn't actually loaded, but just a pointer to something that knows how to fetch it from the database
  - jMock will create a synthetic class that implements an interface for testing purposes

https://softwareengineering.stackexchange.com/questions/123956/why-should-i-use-reflection

When would you use Reflection?

- If you have used any integrated development environment (IDE) to develop a GUI application using drag-and-drop features, you have already used an application that uses reflection in one form or another
- All GUI tools that let you set the properties of a control, say a button, at design time use reflection to get the list of the properties for that control
- Other tools such as class browsers and debuggers also use reflection
- It should be noted that using reflection excessively could slow down the performance of your application
- As an application programmer, you likely will not use reflection much, but the reflection APIs are there if you need them

Python

- Python also has significant support for reflection for introspection and intercession
- Introspection
  - dir(object) – gets all the attributes of an object
  - type checking functions like type(), .__class__, isinstance(), issubclass()
- Intercession
  - Dynamic attribute get and set using setattr()
  - Dynamic methods – calling a method that doesn’t exist in the defined class – using __getattr__

Summary

- Reflection can be used for
  - Introspection
    - Find out about methods of a class
    - Get information about constructors
    - Find out about class fields
  - Intercession
    - Invoke methods by name
    - Create new objects
    - Change values in fields

- For more examples, see:

https://www.oracle.com/technical-resources/articles/java/javareflection.html

- There's also a thorough Java tutorial at:

http://tutorials.jenkov.com/java-reflection/index.html#java-reflection-example

References

[1] Java Language Features: With Modules, Streams, Threads, I/O and Lambda Expressions, Sharan, 2018, Apress

[2] https://docstore.mik.ua/orelly/java-ent/jnut/ch14_01.htm

[3] http://tutorials.jenkov.com/java-reflection/getters-setters.html