Java Reflection

Quiz 2 Revision

Reflection

Reflective programming or *reflection* is the ability of a process to examine, introspect, and modify its own structure and behaviour. (Source: Wikipedia)

- Introspect: A program can observe, and therefore reason about its own state.
- Intercede: A program can modify its execution state or alter its own interpretation or meaning.

Reflection in Java

```
Employee e = new Manager(...);
...
if (e instanceof Manager){
   ...
}
```

- What if we don't know the type that we want to check in advance?
- Suppose we want to write a function to check if two different objects are both instances of the same class?

Reflection in Java . . .

```
public static boolean classequal(Object o1, Object o2){
  ...
  // return true iff o1 and o2 point to objects of same type
  ...
}
```

- We cannot use `instanceof` because we will have to check across all defined classes, which is not a fixed set.
- We cannot use generic type variables becaue `if (o1 instance of T)` is not permitted.

Introspection in Java

- Can extract the class of an object using `getClass()`
- `getClass()` returns an object of type `Class` that encodes class information

```
import java.lang.reflect.*;
class MyReflectionClass{
    public static boolean classequal(Object o1, Object o2){
        Class c1, c2;
        c1 = o1.getClass();
        c2 = o2.getClass();
        return (c1 == c2);
    }
}
```

Using the Class object

- Can create new instances of a class at runtime

```
Class c = obj.getClass();
Object o = c.newInstance();
// Create a new object of same type as obj
```

- Can also get hold of the class object using the name of the class

```
String s = "Manager".
Class c = Class.forName(s);
Object o = c.newInstance();
```

- ..., or, more compactly

```
Object o = Class.forName("Manager").newInstance();
```

The class `Class` ...

- From the `Class` object for class `C`, we can extract details about constructors, methods and fields of `C`
- Constructors, methods and fields themselves have structure
  - Constructors: arguments
  - Methods : arguments and return type
  - All three: modifiers `static`, `private` etc
- Additional classes `Constructor`, `Method`, `Field`
- Use `getConstructors()`, `getMethods()` and `getFields()` to obtain constructors, methods and fields of `C` in an array.

The class `Class` ...

- Extracting information about constructors, methods and fields

```
Class c = obj.getClass();
Constructor[] constructors = c.getConstructors();
Method[] methods = c.getMethods();
Field[] fields = c.getFields();
```

- `Constructor`, `Method`, `Field` in turn have functions to get further details
- Example: Get the list of parameters for each constructor

```
Class c = obj.getClass();
Constructor[] constructors = c.getConstructors();
for (int i = 0; i < constructors.length; i++){
  Class params[] = constructors[i].getParameterTypes();

}
```

Reflection and security

- Can we extract information about private methods, fields,...?
- For private methods, fields:

```
Constructor[] constructors = getDeclaredConstructors();
Method[] methods = getDeclaredMethods();
Field[] fields = getDeclaredFields();
```

- Security issue : Access to private components may be restricted through external security policies
- To be used sparingly