VirginiaTech
Invent the Future

http://web-cat.org

ReflectionSupport: Java Reflection Made Easy

Zalia Shams
Virginia Tech
Ph.D. Student
zalia18@vs.vt.edu

Troubles with Reflection

Reflection is useful for runtime code introspection and manipulation. Hence, it is suitable for automatic code assessment and testing and tools. However, code written using Java Reflection library is bulky, verbose, clunky and unintuitive, because:

- Common tasks (object creation, method invocation etc.) take multiple steps
- Try-catch block around all the sub-steps
- Frequent type casts

ReflectionSupport

- High level abstraction for Java Reflection.
- Open source library provides Static API methods for
  - Object creation(create())
  - Method invocation(invoke())
  - Field manipulation(get(),set())

Why use ?

- No sub-division of tasks
- No Try-catch
- No type casts
- Exceptions thrown by underlying code are unwrapped and passed to user provided handlers
- Diagnostic error reports

Motivating Example

Without Reflection

```
public class ComputerTest
extends TestCase
{
  private Computer computer;
  public void setUp() {
        computer = new
          Computer("xyz", 7, 3.6);
   }

public void testGetProcessor() {
    assertEquals("xyz",
      computer.getProcessor());
 }
//similar testcases
      .....
}
```

With Reflection

```
public class NativeReflectionAPITest extends
TestCase {
      private Object computer;
      public void setUp() {
        try {
            Class compClass =
                        Class.forName("Computer");
            Constructor ctr =
                compClass.getConstructor(
                 String.class, int.class, double.class);
            computer = ctr.newInstance("xyz", 7, 3.6);
          }
          catch (Exception e) {...} ;
        /* here Exception is an umbrella for
          exceptions:
            1. ClassNotFoundException ,
            2. NoSuchMethodException,
            3. InstantiationException,,
            4.SecurityException,
            5. IllegalArgumentException,
            6. IllegalAccessException
             and 7. InvocationTargetException */
      }
      public void testGetProcessor() {
          try {
              Method getProcessorM =
                    computer.getClass().-
                          getMethod("getProcessor");
              assertEquals("xyz", (String)
                    getProcessorM.invoke(computer));
          }
          catch (Exception e) {...}
    }
  ...
}
```

With ReflectionSupport

```
public class ReflectionAPITest
extends TestCase {
      private Object computer;
      public void setUp() {
        computer =
          create("Computer",
                        "xyz",7,3.6);
      }

public void testGetProcessor() {
  assertEquals("xyz",
    invoke(computer,
      String.class,"getProcessor"));
  }
...
}
```

Comparison of ReflectionSupport with Java Reflection API

	Operations Invoked	Source Lines	Required Try/ Catch	Type Casts
Original (no reflection)	12	34	0	0
Java Reflection API	12	93	9	13
ReflectionSupport	12	35	0	4