Advanced Java :: TE D and H
Class Browser – 17th and 18th October 2013
Concepts: Java Reflection

Assignment

- Write a Java program to take a class name and method name as input. Instantiate that class. Check if it contains that method. If not, display an error message. If yes, take appropriate inputs for passing as parameters and call that method
- Write a graphical class browser. It displays complete information about *any* class whose name is supplied in a text box
    - Class.getName(), Class.getSimpleName()
    - Class.getInterfaces()
    - Constructor[] constructors = aClass.getConstructors();
    - Constructor constructor = aClass.getConstructor(new Class[]{String.class});
    - Class[] parameterTypes = constructor.getParameterTypes();
    - Class.getMethods(), method.getName()
    - Class[] parameterTypes = method.getParameterTypes();
    - Class returnType = method.getReturnType();
    - Field[] fields = aClass.getFields();

Basic Concepts

- **RTTI** = Run-Time Type Identification
- Lets you find the exact type of an object when you have only a reference to the base type
- Traditional RTTI assumes you have all types available at compile time and run time
- How it works?
    - There is a Class object for each class that is a part of your program
    - At run time when you want to create an object of that class, JVM first checks to see if the Class object for that class has already been loaded
        - If not, it loads it by finding the .class file with that name
            - If doesn't exist: ClassNotFoundException
        - Note: A java program isn't completely loaded before it begins, unlike many other programming langauges
    - **Class.forName**("Employee");
        - Gives you the reference of corresponding class object
        - Side effect: loads the class
    - Alternatively, Employee.class (called as **class literal**)

        - Also works for primitive types as int.class or boolean.class
    - Class object can be queried for useful run time information
    - Cast operation uses RTTI to ensure that cast is correct
        - Unlike C++ where it simply tells the compiler to treat this object as of the new type
    - if (x **instanceof** Dog) ((Dog)x).bark();
        - This is how you can avoid unsafe downcasts
    - Dynamically creating objects
        - newInstance()
        - throws InstantiationException
    - Dynamic version of instanceof
        - Class.isInstance(object);
    - Subtle differences
        - x.getClass() == Fruit.class
        - x instanceof Fruit
        - Fruit.isInstance(x)
- Reflection allows you to discover class information solely at run time
    - There is a limitation with previous approach → compiler must know about all the classes before hand
    - What if you get an object over the network? Compiler doesn't know anything
    - Two very important uses
        - Component based programming, in which you build projects using RAD in an application builder tool
            - Reflection provides a mechanism to detect available methods and produce the method names
        - Ability to create and execute objects on remote platforms across the network
            - Called RMI
    - java.lang.reflect
        - Field, Method, Constructor
    - .class is still required, but not at compile time
- Another classic example which heavily uses reflection
    - JUnit using annotations

References

- http://tutorials.jenkov.com/java-reflection/index.html
- http://docs.oracle.com/javase/tutorial/reflect/