Dynamic Class Loading

Manuel Oriol - 7th June, 2007

Chair of
Software Engineering

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Default Mechanism

- Classes Loaded on-demand (lazy loading)
- type-safe linkage (ensured at compile-time - verified at load-time)
- User definable

ClassLoaders are Hierarchical

- Access granted to all types defined in your class loader and parents
- if willing to use an instance from code loaded in another class loader, then share a type
- they define namespaces
- they are the only way to garbage collect code

Standard Algorithm

- loadClass()
  - findLoadedClass()
  - loadClass in parent/System
  - findClass
    - defineClass
    - resolveClass

Methods to know

- Constructors
- loadClass
- findLoadedClass
- defineClass
- resolveClass

Example (1/4)

```
import javax.swing.*;

public class MyLoader extends ClassLoader{
    public class MyFileFilter extends javax.swing.filechooser.FileFilter{
            String name;
            public MyFileFilter(String name){
                this.name=name;
            }
            public boolean accept(File f){
                return (f.getName()).endsWith(name+".class");
            }
            public String getDescription(){
                return ".class chooser";
            }

    }
```

Example (2/4)

```
public Class loadClass(String name)throws ClassNotFoundException{
    return loadClass(name,true);
}
public Class loadClass(String name,boolean resolve)
                              throws ClassNotFoundException{
        if (name.startsWith("java.") || name.startsWith("javax.")
                || name.startsWith("sun.")) {
            Class c=findSystemClass(name);
            resolveClass(c);
            return c;
        }else {
       return findClass(name);
    }
}
```

Example 3/4

```
public Class findClass(String name) throws ClassNotFoundException {
        System.out.println("trying to find: "+name);
        Class c = null;
        JFileChooser chooser = new JFileChooser();
        javax.swing.filechooser.FileFilter filter=new MyFileFilter(name);
         chooser.setFileFilter(filter);
         int returnVal = chooser.showOpenDialog(null);
         try{
         if(returnVal == JFileChooser.APPROVE_OPTION) {
                File file = chooser.getSelectedFile();
                FileReader fr=new FileReader(file);
                long l=file.length();
                char[] cBuf=new char[(int)l+1];
                fr.read(cBuf,0,(int)l);
                byte[] bBuf=(new String(cBuf)).getBytes();
                c=defineClass(name,bBuf,0,(int)l);
                resolveClass(c);
         }} catch (Exception e){
              System.out.println("Class "+name+" not found.");
              throw new ClassNotFoundException();}
     return c;
}
```

Example (4/4)

```
    public static void main(String[] args) throws Exception{
        ClassLoader cl = new MyLoader();
        Class t=cl.loadClass("T");
        t.newInstance();
    }
}
```

What happens with

```
import java.io.*;

public class T {
   public T(){
      System.out.println("T instance created");
      new T2();
   }

}
```

```
import java.io.*;

public class T2 {
   T2(){
      System.out.println("T2 instance created");
   }

}
```

To go further...

"Dynamic Control of Adaptive Parameters in Evolutionary Programming", by Liang and Bracha