Dynamisches Laden von Klassen

- Die Klasse class und die Klassen des Packages java.lang.reflect erlauben Reflection und Intospection.
- Klassen können zur Laufzeit geladen, untersucht und instanziert werden. Zu den dann erzeugten Objekten können Methoden aufgerufen werden.
- Als Beispiel werde ein Programm betrachtet, das Java-Quelltext in einer Datei erzeugt, einen Compileraufruf generiert und danach die neu entstandene Klasse lädt und eine Methode dazu aufruft.

Laden der Klasse

```
Class X=Class.forName(Name);
```

Instanzieren der Klasse

```
Calculate C=(Calculate)(X.newInstance());
```

Metodenaufruf

```
System.out.println("Y(x)="+C.fVonX(x));
```

Caclulate ist ein Interface

```
public interface Calculate
{
  public double fVonX(double x);
}
```

Beispiel: Ausdrucksberechnung

- Ein Programm solle in der Lage sein, beliebeige Ausrücke f(x) zu berechnen.
- Lösungsansatz 1: Ausdrucksberechner mit Verfahren des rekursiven Abstiegs programmieren.
- Lösungsansatz 2: Verwendung von Jcup/Jlex um einen Interpreter für Ausdrücke generieren zu lassen
- Lösungsansatz 3: Generieren eines Javaquelltextes einer Klasse mit der Methode fvonx, Übersetzung, Laden, Instanzieren der Klasse und Ausführung der Methode fvonx.

Erzeugen des Quelltextes

- Class.forName funktioniert für jede zu ladende Klasse nur ein mal.
- Workaround: fortlaufender Dateiname

```
// Hier wird der Name zusammengebaut
Name="dcalc"+Count;

FileWriter O=new FileWriter(Name+".java");

O.write("class "+Name+" implements Calculate\n");
O.write("{public double fVonX(double x)“);
O.write("{return "+Command+";}}");
O.close();
```

Compilieren

```
Process p = Runtime.getRuntime()
            .exec("javac "+Name+".java");
System.out.println("Compilation of "+Name+" started");
p.waitFor();

String line;
BufferedReader input =
    new BufferedReader(new InputStreamReader
                                    (p.getErrorStream()));
 while ((line = input.readLine()) != null)
 {
   System.out.println(line);
 }
 input.close();
```

Laden und instanzieren

```
void calculate(double x)
{
  /* Durch Einfuehrung der Funktion
     calculate verliert das Objekt
     Class X nach jeder Ausfuehrung
     sein Leben wird also jedesmal
     neu angelegt. */
  try
  {
    Class X=Class.forName(Name);
    Calculate C=(Calculate)(X.newInstance());
    System.out.println("Y(x)="+C.fVonX(x));
  }
  catch(Exception e)
       {System.out.println("Exception " + e);}
}
```

Der Classloader

- Jede Klasse wird nur einmalig geladen.
- Ausweg, Benutzung eines anderen Classloaders: URLClassLoader
- Wird mit einem Array von URLs instanziert.
- Erzeugte Klasse muss in einem Unterverzeichnis liegen, davon URL kreieren, damit URKClassLoader instanzieren, damit Klasse laden - fertig.

URL bauen

```
// Get the directory (URL) of the reloadable class
URL[] urls = null;
try
{
 // Convert the file object to a URL
 File dir = new File(System.getProperty("user.dir")
          +File.separator+"XFX"+File.separator);
 URL url = dir.toURI().toURL();
 System.out.println("URL:"+url);
 urls = new URL[]{url};

}
catch (MalformedURLException e)
{
        System.out.println(e);
}
```

Classsloader instanzieren und rechnen

```
try
{
 // Create a new class loader with the directory
 ClassLoader cl = new URLClassLoader(urls); //java.net

 // Load in the class
 Class cls = cl.loadClass("dcalc");

 // Create a new instance of the new class
 C = (Calculate)cls.newInstance();
}
catch (IllegalAccessException e) {System.out.println(e);}
catch (InstantiationException e) {System.out.println(e);}
catch (ClassNotFoundException e) {System.out.println(e);}
System.out.println("Y="+Command);
System.out.println("Y(x)=Y("+x+")="+C.fVonX(x));
```