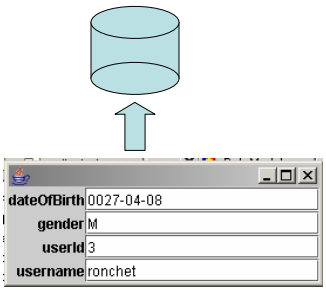
Java introspection

java.reflect.*

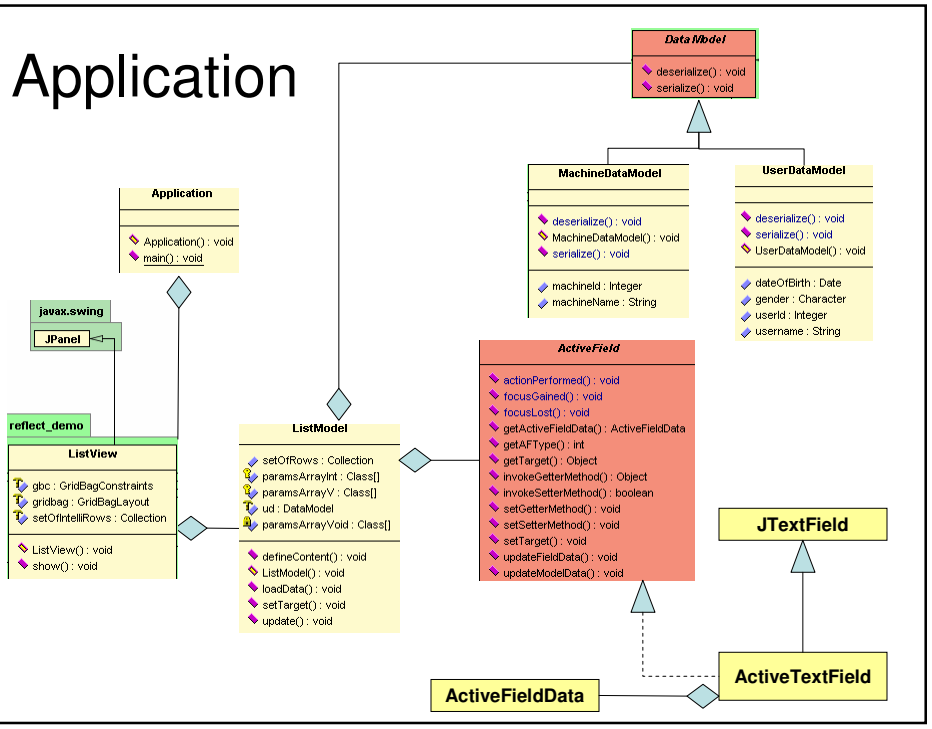
Declarative Programming

```
 private Date dateOfBirth = null;
  private Integer userId = null;
  private String username = null;
  private Character gender = null;

// Methods
  public Date getDateOfBirth() {
    System.out.println("reading Date");
    return dateOfBirth;
  }

  public void setDateOfBirth(Date fk) {
    System.out.println("writing dateOfBirth");
    dateOfBirth = fk;
    this.serialize();
  }

...
```

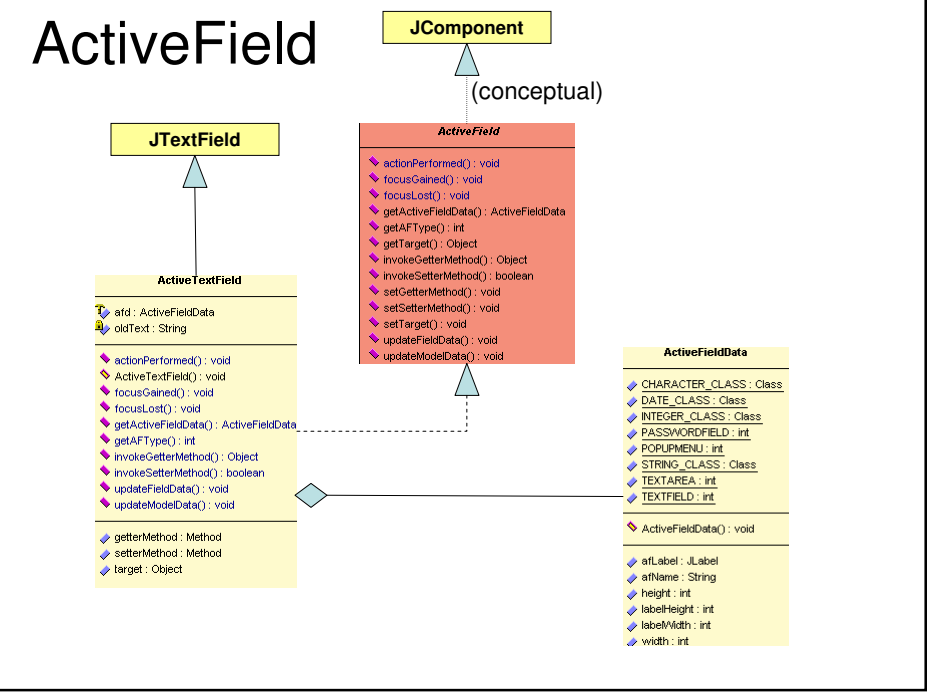



```
...
```

Application



ActiveField



API

- java.lang.Class
- java.lang.reflect.*

Simple Dynamic Execution

```
package reflect;

import java.io.*;
import java.lang.reflect.*;

public class Dynamic {

 void printMethods(Class c) {
  Method methods[]=c.getDeclaredMethods();
    for (int i=0; i<methods.length; i++) {
    System.out.print(i+" "+methods[i].getReturnType());
    System.out.print(" "+methods[i].getName()+"(");
    System.out.println(")");
  }
}
```

1

```
    public static void main(String[] args) {
   Dynamic d=new Dynamic();
// read and create factory class
   System.out.println("Dammi il nome
     della classe
     (incluso il package)");
   String className=Cum.readString();
   Class c=null;
   try {
    System.out.println("Creating Class "
      +className);
    c=Class.forName(className);
   } catch (Exception e) {
    Cum.printError(e);
   }
```

2

```
// create instance
   Object o=null;
   try {
    o=c.newInstance();
   } catch (Exception e) {
    Cum.printError(e);
   }
   System.out.println("ToString gives:"
     +o.toString());
```

```
// print methods of the class
  d.printMethods(c);

// read and invoke a method
  System.out.println("Dammi il nome del metodo da eseguire");
  String methodName=Cum.readString();
  try {
   Method m=c.getMethod(methodName,null);
   Object rv=m.invoke(o,null);
   if (rv != null) System.out.println("Returned value: "+rv.toString());
  } catch (Exception e) {
   Cum.printError(e);
   }
  System.out.println("ToString gives:"+o.toString());

// wait an input to close the program
  Cum.readString();
 }
}
```

3

Utilities

```
class Cum {
//Common Utilities Marco
//================================
// forma compatta per sleep
public static void aspetta(int secs) {
      try {
        Thread.sleep(secs*1000);
      } catch (InterruptedException i) {}
  }
//================================
// forma compatta per sleep, con msg
public static void aspetta(int secs, String s) {
      System.out.println(s);
      try {
        Thread.sleep(secs*1000);
      } catch (InterruptedException i) {}
  }
```

```
// input di Stringa
public static String readString() {
    char c;
    StringBuffer s=new StringBuffer(0);
    try {
      Reader in = new InputStreamReader(System.in);
      while ( ( (c=(char)in.read()) != '\n') && (c != '\r') ) {
        s.append(c);
      }
    } catch (Exception e) {
      //System.out.println("Errore: "+e.toString());
      printError(e);
    }
    String st = s.toString();
    return st;
  }
//====================================
 // stampa di errore, con pausa
public static void printError(Exception e) {
      System.out.println("Errore: "+e.toString());
      readString();
      System.exit(1);
  }
}
```

Ricava tutte le info di una classe - 1

```
package reflect2;
import java.lang.reflect.*;
import java.io.*;
/** A program that displays a class synopsis for the named class. */
public class ShowClass {
  /** The main method.  Print info about the named class. */
  public static void main(String[] args) {
    Class c = null;
    String className=Cum.readString();
    System.out.println(className);
    try {
      System.out.println("Creating Class "+className);
      c=Class.forName(className);
    } catch (Exception e) {
      Cum.printError(e);
    }
    print_class(c);
    Cum.readString();
  }
```

```
//================================================
  /** Display the modifiers, name, superclass, and interfaces of a class
   *  or interface. Then go and list all constructors, fields, and methods. */
  public static void print_class(Class c)
  {
    // Print modifiers, type (class or interface), name, and superclass.
    if (c.isInterface()) {
      // The modifiers will include the "interface" keyword here...
      System.out.print(Modifier.toString(c.getModifiers()) +  c.getName());
    }
    else
      System.out.print(Modifier.toString(c.getModifiers()) + " class " +
                       c.getName() +
                       " extends " + c.getSuperclass().getName());
    // Print interfaces or super-interfaces of the class or interface.
    Class[] interfaces = c.getInterfaces();
    if ((interfaces != null) && (interfaces.length > 0)) {
      if (c.isInterface()) System.out.println(" extends ");
      else System.out.print(" implements ");
      for(int i = 0; i < interfaces.length; i++) {
        if (i > 0) System.out.print(", ");
        System.out.print(interfaces[i].getName());
      }
    }
```

Ricava tutte le info di una classe-2

```
    System.out.println(" {");           // Begin class member listing.
    // Now look up and display the members of the class.
    System.out.println(" // Constructors");
    Constructor[] constructors = c.getDeclaredConstructors();
    for(int i = 0; i < constructors.length; i++)  // Display constructors.
      print_method_or_constructor(constructors[i]);
    System.out.println(" // Fields");
    Field[] fields = c.getDeclaredFields();       // Look up fields.
    for(int i = 0; i < fields.length; i++)        // Display them.
      print_field(fields[i]);
    System.out.println(" // Methods");
    Method[] methods = c.getDeclaredMethods(); // Look up methods.
    for(int i = 0; i < methods.length; i++)           // Display them.
      print_method_or_constructor(methods[i]);
    System.out.println("}");            // End class member listing.
  }
//================================================
  /** Return the name of an interface or primitive type,
   *  handling arrays. */
  public static String typename(Class t) {
    String brackets = "";
    while(t.isArray()) {
      brackets += "[]";
      t = t.getComponentType();
    }
    return t.getName() + brackets;
  }
```

```
//================================================
  /** Return a string version of modifiers, handling spaces nicely. */
  public static String modifiers(int m) {
    if (m == 0) return "";
    else return Modifier.toString(m) + " ";
  }
  /** Print the modifiers, type, and name of a field. */
  public static void print_field(Field f) {
    System.out.println("  " +
                       modifiers(f.getModifiers()) +
                       typename(f.getType()) + " " + f.getName() + ";");
  }
```

una classe-3

-

```
  /** Print the modifiers, return type, name, parameter types,
   * and exception type of a method or constructor.  Note the use of
   * the Member interface to allow this method to work with both
   * Method and Constructor objects. */
  public static void print_method_or_constructor(Member member) {
    Class returntype=null, parameters[], exceptions[];
    if (member instanceof Method) {
      Method m = (Method) member;
      returntype = m.getReturnType();
      parameters = m.getParameterTypes();
      exceptions = m.getExceptionTypes();
    } else {
      Constructor c = (Constructor) member;
      parameters = c.getParameterTypes();
      exceptions = c.getExceptionTypes();
    }
    System.out.print("  " + modifiers(member.getModifiers()) +
                     ((returntype!=null)? typename(returntype)+" " : "") +
                     member.getName() + "(");
    for(int i = 0; i < parameters.length; i++) {
      if (i > 0) System.out.print(", ");
      System.out.print(typename(parameters[i]));
    }
    System.out.print(")");
    if (exceptions.length > 0) System.out.print(" throws ");
    for(int i = 0; i < exceptions.length; i++) {
      if (i > 0) System.out.print(", ");
      System.out.print(typename(exceptions[i]));
    }
    System.out.println(";");
  }
}
```