Class

In Java, the runtime environment knows the runtime type info for all objects.

- True polymorphism

```java
Object ob = new Foo();

We still know that ob is an object of type Foo.
```

```java
Class c1 = ob.getClass();

Class class contains info about the class of an object.

Ex: ob.getClass().getName() gives the name of the class.
```

```
Alt:
String className = "Foo";
Class c2 = Class.forName(className);
```

```java
Class c3 = Foo.class;
Class c4 = int.class;
```

Creating instances of objects:
```java
String s = "Foo";
Object m = Class.forName(s).newInstance();
```
Reflection

A program that can analyze the capabilities of classes is called **reflective**.

java.lang.reflect

Using the reflection API, one can get information about a class:

- Field: data fields of the class
- Method: methods of the class
- Constructor: constructors of the class

```java
Class c1 = myObject.getClass();
Method[] methods = c1.getMethods();
Field[] fields = c1.getFields();
```

A program can analyze any class that the Java interpreter can load, not just the classes available when the program was compiled.
Getting and Setting Data

Employee harry = new Employee("Harry Hacker", 35000, 
    new Date(10, 1, 1989));
// Get the class object representing Employee
Class c1 = harry.getClass();
// Get the name field of the Employee class
Field f = c1.getField("name");
// Get the value of the name field from object harry
Object v = f.get(harry);

There are issues with permissions. By default, only public fields are available, but there are ways around this.

Not only can you get a field value, you can set it!
Generic Array Example

```java
static Object[] arrayGrow(Object[] a) {
    int newLength = a.length * 11 / 10 + 10;
    Object[] newArray = new Object[newLength];
    System.arraycopy(a, 0, newArray, 0, a.length);
    return newArray;
}

This is an array of Objects, not Employees!

static Object[] arrayGrow(Object[] a) {
    Class c1 = a.getClass();
    if (!c1.isArray()) return null;
    Class compType = a.getClass().getComponentType();
    int length = Array.getLength(a);
    int newLength = a.length * 11 / 10 + 10;
    Object newArray = Array.newInstance(compType,
                     newLength);
    System.arraycopy(a, 0, newArray, 0, a.length);
    return newArray;
}
```
Introspection

Builds on reflection to get information on Beans...
- From the BeanInfo of a class, or
- From the JavaBean naming conventions.

Introspection is normally done by the Bean Environment, not by beans.
- But, there may be reasons for a bean to examine other beans.

Example: Want one bean to have increment/decrement buttons that modify the value in the field of another bean on-the-fly.
- Look at BeanInfo properties of other bean.
- Determine which properties are of type int.
- Provide a selection list of these properties to user.
- Use get/set from reflection package to modify field contents.