Working with a Data Access Layer

Object Oriented Programming
2016375 - 5

Camilo López

Outline

- Introduction
- The Class File
- Reading from a file
- Writing to a file

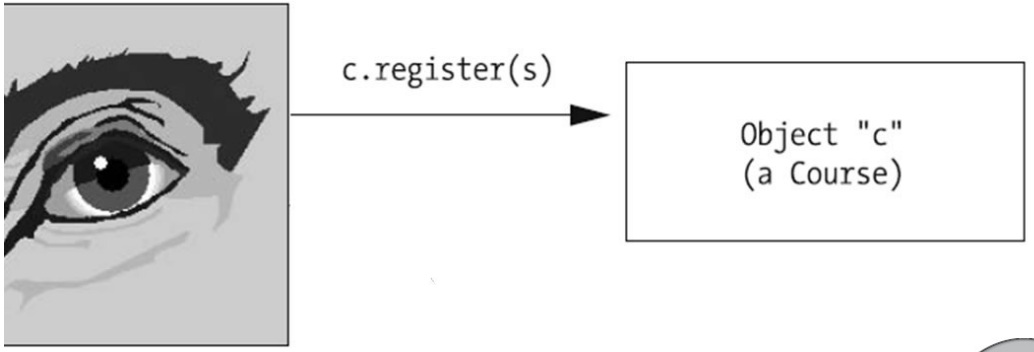
Introduction

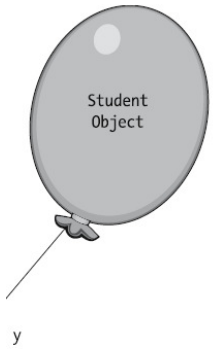
- What have we done so far?

There's a triggering event
*Any request from client code

Introduction

- What have we done so far?

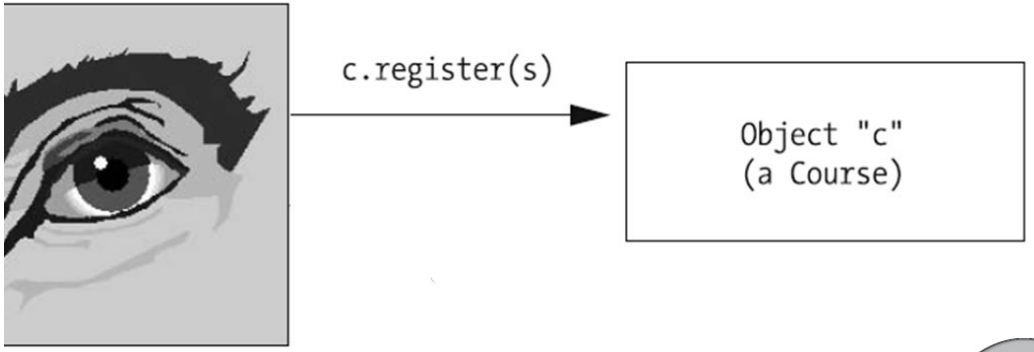


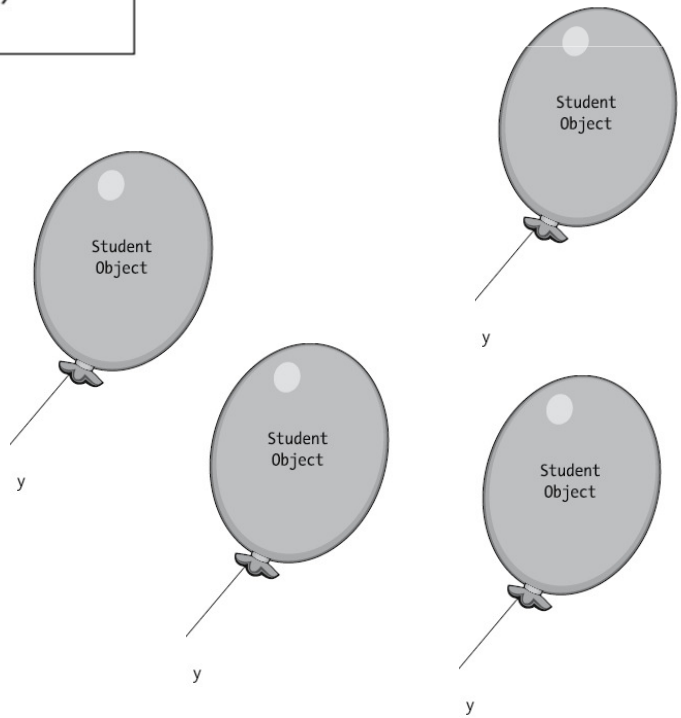


An object is created...

Introduction

- What have we done so far?

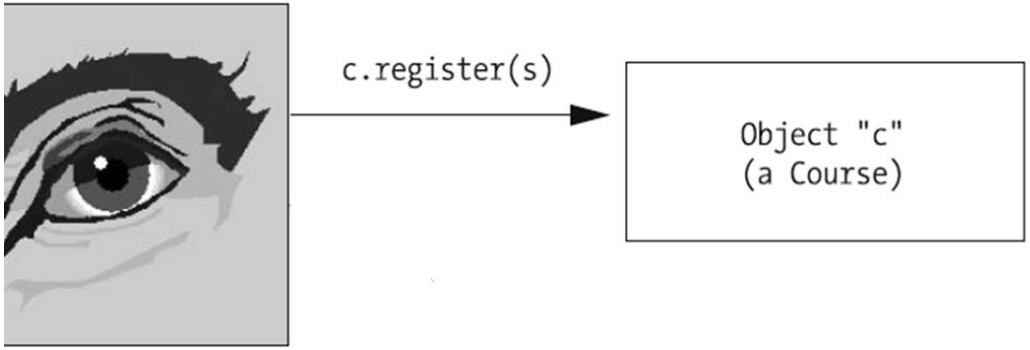


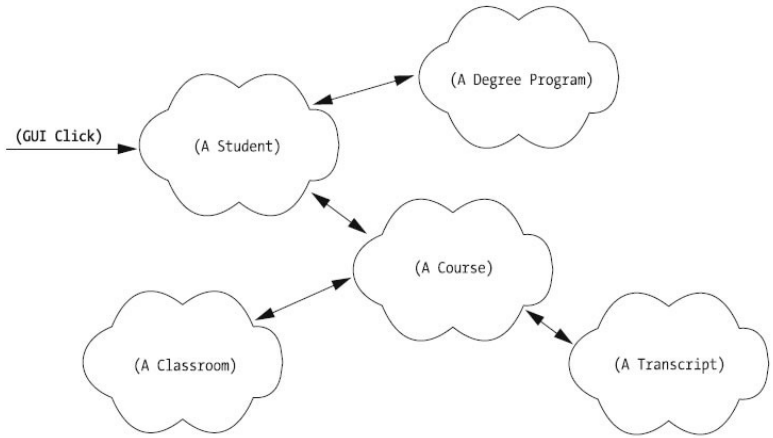


An object is created...
Maybe several objects

Introduction

- What have we done so far?

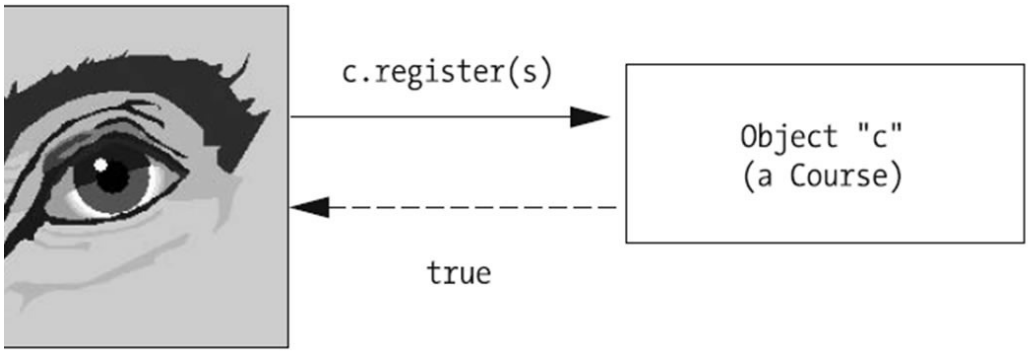




There's a communication between the objects...

Introduction

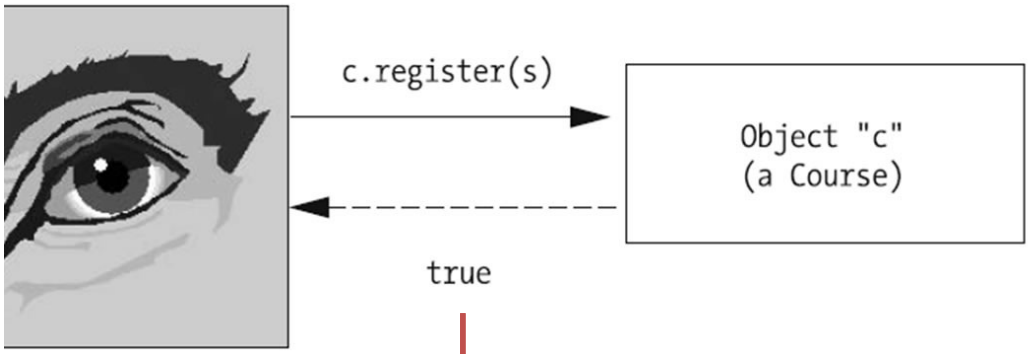
- What have we done so far?



...In order to comply with the request

Introduction

- What have we done so far?

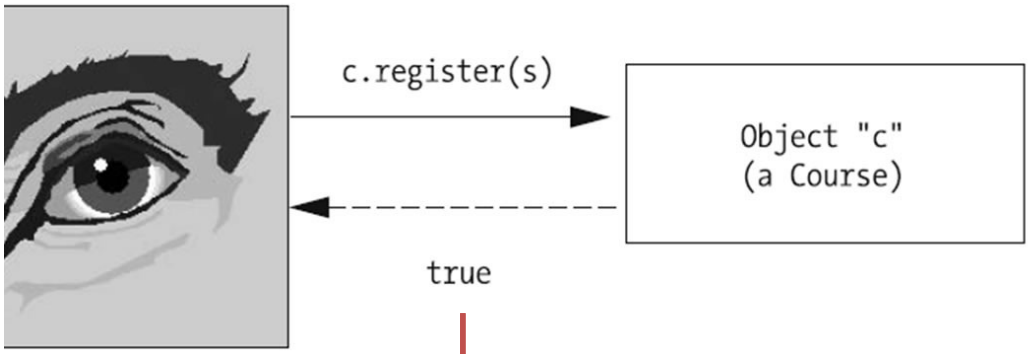


where do you get the necessary information?

...In order to comply with the request

Introduction

- What have we done so far?



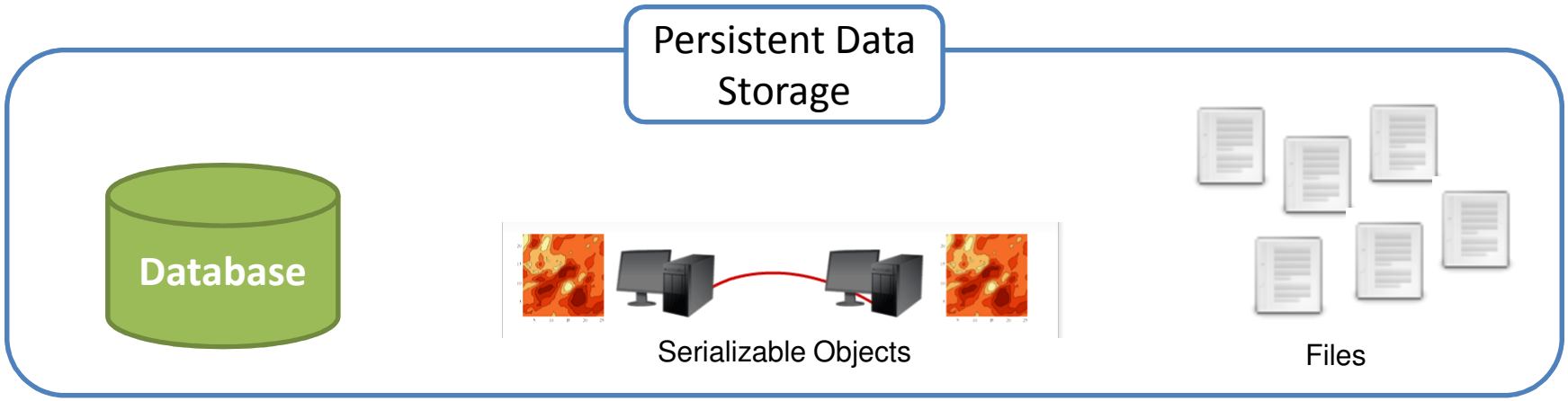
where do you get the necessary information?

...In order to comply with the request

what do you do with the answer?

Introduction

- The application provides no means of remembering the state of the objects from one invocation of the application to the next.
  - Whenever we run a Java application, all objects that we instantiate reside in memory allocated to the JVM. When such an application terminates, all of the JVM's memory is released back to the operating system, and the internal states of all of the objects created by the application are forgotten, unless they have been **persisted** (saved).



The Class File

- The Class File is particularly **useful for retrieving information about files or directories from disk**. Objects of class File do not open files or provide any file-processing capabilities.
  - Class File provides four constructors.

```
File fileName = new File(String path);
```

Creates a new File instance by converting the given pathname string into an abstract pathname

```
File f = new File("C:\file.txt");
File names = new File("newFile.dat");
```

For more info:

http://java.sun.com/javase/6/docs/api/java/io/File.html

The Class File

Method	Description
`boolean canRead()`	Returns `true` if a file is readable by the current application; `false` otherwise.
`boolean canWrite()`	Returns `true` if a file is writable by the current application; `false` otherwise.
`boolean exists()`	Returns `TRue` if the name specified as the argument to the `File` constructor is a file or directory in the specified path; `false` otherwise.
`boolean isFile()`	Returns `true` if the name specified as the argument to the `File` constructor is a file; `false` otherwise.
`boolean isDirectory()`	Returns `true` if the name specified as the argument to the `File` constructor is a directory; `false` otherwise.

The Class File

Method	Description
`String getAbsolutePath()`	Returns a string with the absolute path of the file or directory.
`boolean isAbsolute()`	Returns `TRue` if the arguments specified to the `File` constructor indicate an absolute path to a file or directory; `false` otherwise.
`String getName()`	Returns a string with the name of the file or directory.
`String getPath()`	Returns a string with the path of the file or directory.
`String getParent()`	Returns a string with the parent directory of the file or directory (i.e., the directory in which the file or directory can be found).

Reading from a File

```
    FileReader fr = new FileReader(nameOfFileToBeReadFrom);
    BufferedReader bIn = new BufferedReader(fr);

    String line = bIn.readLine();

    while (line != null) {
        // Process the most recently read line however we'd like ...
        line = bIn.readLine();
    }

    bIn.close();
}
```

Instantiate a FileReader, and pass it into a BufferedReader.

Reading from a File

```
    FileReader fr = new FileReader(nameOfFileToBeReadFrom);
    BufferedReader bIn = new BufferedReader(fr);

    String line = bIn.readLine();

    while (line != null) {
        // Process the most recently read line however we'd like ...
        line = bIn.readLine();
    }

    bIn.close();
}
```

Read the first line from the file.

Reading from a File

```
    FileReader fr = new FileReader(nameOfFileToBeReadFrom);
    BufferedReader bIn = new BufferedReader(fr);

    String line = bIn.readLine();

    while (line != null) {
        // Process the most recently read line however we'd like ...
        line = bIn.readLine();
    }

    bIn.close();
}
```

As long as the end of the file hasn't been reached ...

Reading from a File

```
    FileReader fr = new FileReader(nameOfFileToBeReadFrom);
    BufferedReader bIn = new BufferedReader(fr);

    String line = bIn.readLine();

    while (line != null) {
        // Process the most recently read line however we'd like ...
        line = bIn.readLine();
    }

    bIn.close();
}
```

Close the BufferedReader, which automatically closes the encapsulated FileReader, as well.

Writing to a File

```
    FileOutputStream fos = new FileOutputStream(nameOfFileToBeWrittenTo);
    PrintWriter pw = new PrintWriter(fos);

    while (we still have more data to output) {
        pw.println(whatever data we wish to output);
    }

    pw.close();
}
```

Instantiate a FileOutputStream, and pass it into a PrintWriter.

Writing to a File

```
    FileOutputStream fos = new FileOutputStream(nameOfFileToBeWrittenTo);
    PrintWriter pw = new PrintWriter(fos);

    while (we still have more data to output) {
        pw.println(whatever data we wish to output);
    }

    pw.close();
}
```

FileOutputStream(String filename, boolean append)

Instantiate a FileOutputStream, and pass it into a PrintWriter.

Writing to a File

```
    FileOutputStream fos = new FileOutputStream(nameOfFileToBeWrittenTo);
    PrintWriter pw = new PrintWriter(fos);

    while (we still have more data to output) {
        pw.println(whatever data we wish to output);
    }

    pw.close();
}
```

Close the PrintWriter, which automatically closes the encapsulated FileOutputStream, as well.

References

- J. Barker, *Beginning Java Objects: From Concepts To Code, Second Edition*, Apress, 2005.
- H.M. Deitel and P.J. Deitel, Java How to Program: Early Objects Version, Prentice Hall, 2009.