CSE1720

Delegation Concepts (Ch 2)

Output (sec 2.2.5)

- Output to the console
- Output to a file (later… section 5.3.2)

- Instead of
  ```
  System.out.println("Hi");
  ```
- Use:
  ```
  PrintStream output = System.out;
  output.println("Hi");
  ```

Input (sec 2.2.5)

- Output from the console
- Input from a file (later… section 5.3.2)

- Use:
  ```
  Scanner input = new
  Scanner(System.in);
  int x = input.nextInt();
  ```

Ready-Made I/O Components

Use this template as a starting point for all your programs in this course:

```
import java.util.Scanner;
import java.io.PrintStream;

public class Template
{
  public static void main(String[] args)
  {
    Scanner input = new Scanner(System.in);
    PrintStream output = System.out;
    …
    // use input.nextInt/Double for input
    // use output.println/print for output
    …
  }
}
```

2.1 Computing Paradigms

```
import java.lang.System;

public class Area
{
   public static void main(String[] args)
   {
      int width = 8;
      int height = 3;
      int area = width * height;
      System.out.println(area);
   }
}
```

The code inside the rectangle computes the area of a circle. It handles both storage (of data) and computation (of area). Let us explore delegating one or both of these tasks.

2.1.1 Procedural Paradigm

Keep storage but delegate computation to a class:

```
int width = 8;
int height = 3;
int area = Rectangle1.computeArea(width,  height);
```

- A method belongs to a class. It performs an action, and hence, its name is a verb (e.g., `computeArea()`) or a complete predicate (e.g., `isEnabled()`).
- The method name must be followed by a pair of parenthesis with any parameters needed sandwiched in between.
- The method name together with the types of its parameters make up the method signature. It is unique per class.
- The method's action culminates in a return. It can be void.
- Invocation syntax: **class_name.method(…)**. It is like dialing the phone number of a company followed by someone's extension.

2.1.2 Modular Paradigm

Delegate both storage and computation to a class:

```
Rectangle2.width = 8;
Rectangle2.height = 3;
int area = Rectangle2.getArea();
```

- An attribute belongs to a class. It holds data, and hence, its name is a noun (width). It has a type.
- Java treats attributes like variables except you do not declare them in your program (their class takes care of that) and the notion of scope does not apply to them.
- The attribute name is unique per class.
- Access syntax: **class_name.attribute**.
- Because the class name appears before the dot, we say that you invoke a method, or access an attribute, **on the class**.

2.1.3 Object-Oriented Paradigm

Delegate both to an instance of a class:

```
Rectangle3 r = new Rectangle3();
r.width = 8;
r.height = 3;
int area = r.getArea();
```

- Create an instance (a.k.a object) of a class that can handle storage and computation and work with the instance as if it is a module.
- The instance has a name, r, known as the **object reference**.
- The attributes are accessed, and the methods are invoked, **on the instance**, not on the class.
- Think of the object (or instance) as a copy of the original class.
- Each object can store different values in its attributes; these values are known as the state of the object.

Case Study 2.2.4: the JDK

Top-level packages

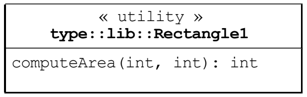
java.awt	Provides support for drawing graphics. AWT = Abstract Windowing Toolkit
java.beans	Provide support for Java Beans.
java.io	Provides support for f ile and other I/O operations.
java.lang	Provides the fundamental Java classes. This package is auto -imported by the compiler.
java.math	Provides support for arbitrary -precision arithmetic
java.net	Provides su pport for network access.
java.rmi	Provides support for RMI. RMI = Remote Method Invocation
java.security	Provides support for the security framework.

java.sql	Provides support for databases access over JDBC
	JDBC = Java Database Connectivity,
	SQL = Structured Query Language

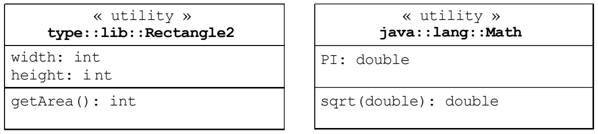
java.text	Provides formatting for text, dates, and numbers.
java.util	Miscellaneous utility classes including JCF. JCF = Java Collection Framework
javax.crypto	Provides support for cryptographic operations .
javax.servlet	Provides support for servlet and JSP development. JSP = Java Server Pages
javax.swing	Provides support for GUI development. GUI = Graphical User Interface
javax.xml	Provides support for XML processing. XML = eXtensible Markup Language

UML (Unified Modeling Language)

The class diagram of a procedural class:

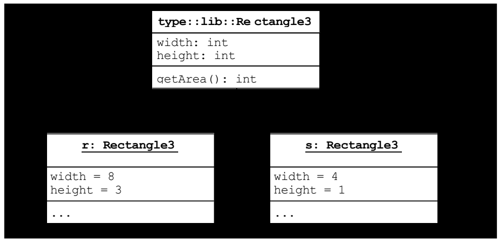


The class diagrams of two modular classes:

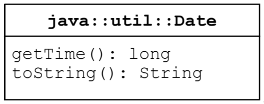


UML (Non-Utility Classes)

The class diagram of an object oriented class along with the object diagrams of two instances of it:



The class diagrams of an object-oriented class in the Java standard library



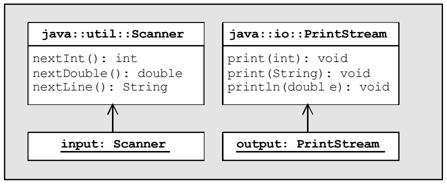
2.2.5 Ready-Made I/O Components

Keyboard Input:

```
Scanner input = new Scanner(System.in);
int width = input.nextInt();
```

Screen Output:

```
PrintStream output = System.out;
output.print(width);
```



2.2.1 Application Architecture

- A Java application consists of several cooperating classes. One of the classes starts the application, and is known as the main class. The other classes are known as helpers or components.
- The main class for a desktop application (as opposed to an applet or servlet) is known as an app. It must have a method with the following header:

```
public static void main(String[] args)
```

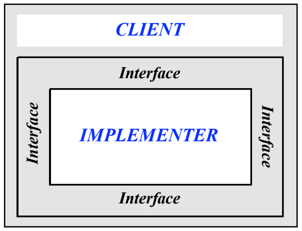
- The main class delegates to components. And as more ready-made components become available, application development will reduce to developing the main class.

2.2.2 The Client View

- The client is the developer of the main class. The implementer is the developer of a component.
- The client understands the big picture, the purpose of the application. The implementer focuses only on the inner details of one component.
- The client knows how to shop for components and how to read their specs; i.e. knows what each one does but not how it does it.
- This course focuses on being a client. It prepares you to write applications using components that are already available.
- Separation of concerns means the client and the implementer share info on a need-to-know basis.

The Client View

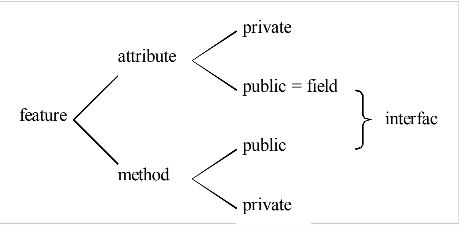
- Given a component, the client does not care what is inside it, only what it does. This is known as its interface or API (application programming interface).
- The class of a component thus encapsulates it. An attempt to look inside is breaking the encapsulation.



The Client View

A class is made up of features. A feature is an attribute or a method. The class of a component classifies each feature as either public or private depending, respectively, on whether the client needs or does not need to know about it.

The API (interface) of a component lists only the headers of its public methods and the declarations of its public attributes (a.k.a. fields).



2.3.1 Risk Mitigation
by Early Exposure

If you are not sure about something during software development, confront it as early as possible. Making changes later is more difficulty than doing so now.

For example, the Java compiler turns a potential logic error (like assigning a real value to an int variable) to a compile-time error. The risk of truncating the real value is exposed early.

2.3.3 Contracts

Each method in a component comes with a contract that spells out the responsibilities of the client and the implementer.

The client must supply parameters that satisfy the precondition of the method.

The implementer must supply a return that satisfy the postcondition of the method.

Liability:

•if pre=false, the client is at fault,

•if pre=true and post=false, then the implementer is at fault.

•If pre=post=true then everything is OK.

Note: if a method has pre=true then its client does not have to ensure anything.

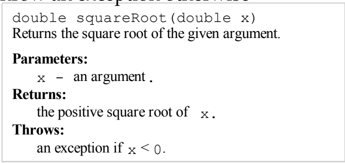
Contracts

Methods in the Java standard library specify their pre and post as follows:

- pre is always assumed to be true unless stated otherwise

- post is specified under Returns and Throws and can be assumed to be true

Example:
This contract specifies pre=true (i.e. no condition on the parameter). The post states that the method will return the square root if x is non-negative and will throw an exception otherwise



`double squareRoot(double x)`
Returns the square root of the given argument.

Parameters:
`x` - an argument.

Returns:
the positive square root of `x`.

Throws:
an exception if `x` < `0`.