CSE 1710

Lecture 6
The Client View

The assigned reading was:

- **The Client View**
  - sec 2.2.2, pp. 60-64
- **Post-Compilation Errors**
  - sec 2.2.3, pp. 64-65
- **Java Standard Library**
  - sec 2.2.4, pp. 66-68
- **Readymade I/O**
  - sec 2.2.5, pp. 68-70

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)
```

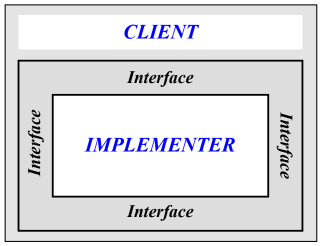
- The main class delegates to components. And as more ready-made components become available, application development will reduce to developing the main class.

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

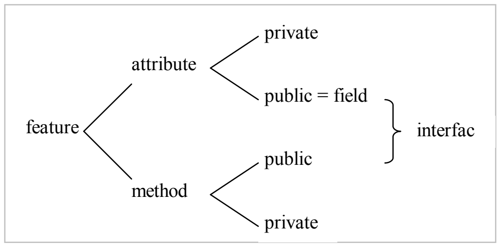
- Given a component, the client does not care what is inside it, only what it does. This is known as its interface or API (application programming interface).
- The class of a component thus encapsulates it. An attempt to look inside is breaking the encapsulation.



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).



2.2.3 Post-Compilation Errors

E

- Launch an editor and write the program
- Save it as Area.java

Post-Compilation Errors

E

- Launch an editor and write the program
- Save it as Area.java

C

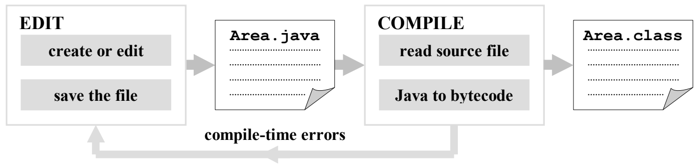
- Launch a console
- Compile by issuing: javac Area.java
- Barring errors, this generates Area.class

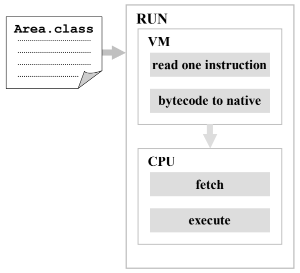
Post-Compilation Errors

E
- Launch an editor and write the program
- Save it as Area.java

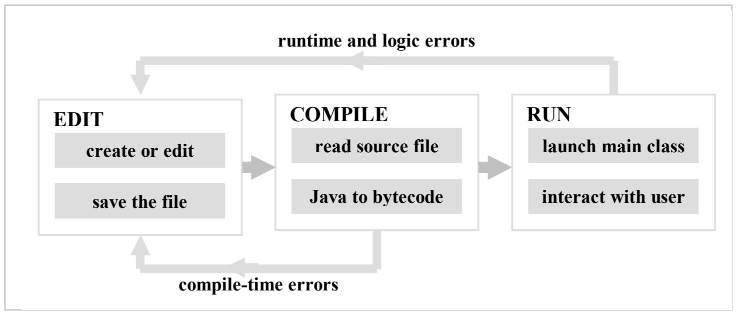
C
- Launch a console
- Compile by issuing: javac Area.java
- Barring errors, this generates Area.class

R
- Run Area.class by issuing: java Area
- Enjoy!



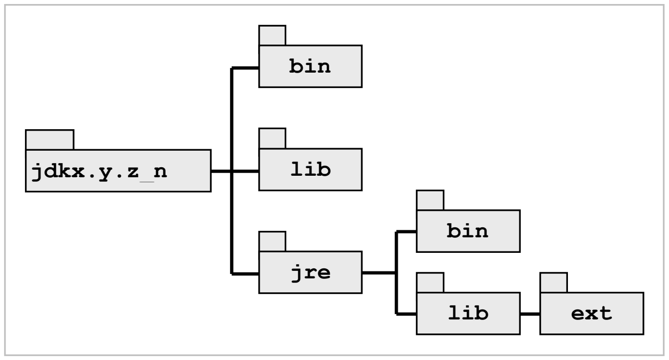


Post-Compilation Errors



2.2.4 Case Study: the JDK

Directory structure:



Case Study: 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 file 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 support 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

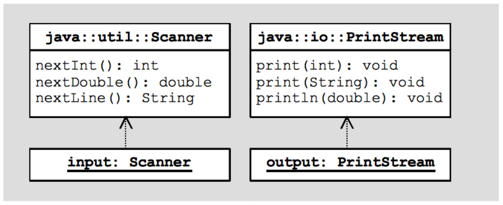
2.2.5 Ready-Made I/O Components

Keyboard Input:

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

Screen Output:

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



Ready-Made I/O Components

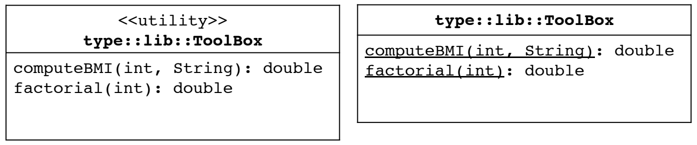
Use this template as a starting point for all your programs in this course:

```
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
        ...
    }
}
```

UML (Unified Modeling Language)

Consider the following UML class diagrams:

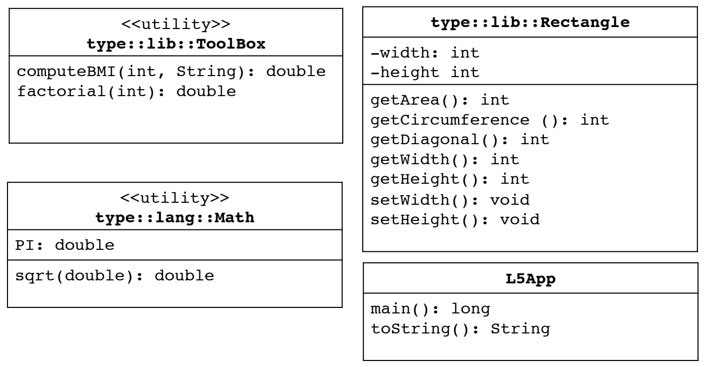


underlined method name indicates the method is static

recall: a utility class is a class that cannot be instantiated

UML (Unified Modeling Language)

Consider the following UML class diagrams:



<<utility>>
type::lib::ToolBox
computeBMI(int, String): double
factorial(int): double
<<utility>>
type::lang::Math
PI: double
sqrt(double): double
type::lib::Rectangle
-width: int
-height int
getArea(): int
getCircumference (): int
getDiagonal(): int
getWidth(): int
getHeight(): int
setWidth(): void
setHeight(): void
L5App
main(): long
toString(): String