CSE 1710

Lecture 4

Delegation

The Notion of Delegation (§2.1.4)

- Delegation in a specific version of *abstraction*
- Abstraction:
  - the abstract description of a process (or thing) is a version in which the details have been removed and replaced with a simpler substitute;
  - that which is filtered away determines the type of abstraction

Examples of types of abstraction

- Abstraction by parameterization
- Abstraction by delegation

Abstraction by Parameterization

- this applies to physical objects (in a simple case)
  - what to do: see the object's properties as detailed versions of the object's property parameters
  - if you filter away the property values from the property parameters, you get *abstraction by parameterization*
    - detailed version: this car is a BMW, it has 15" rims and is silver
    - abstract version: a car is a thing has a maker, has a wheel size and has a colour
    - even more abstract version: a vehicle has a means of locomotion and a means of steering
      - this is also true of tanks and trains, but the means of locomotion and the means of steering are different in these cases

Abstraction by Delegation

- applies to processes that perform tasks:
  - what to do: see the "how" as a detailed version of the "what" when it comes to doing things
  - if you filter away the "how" from the "what", you get *abstraction by (task) delegation*
    - detailed version: grow grain, grind grain, combine with yeast to make dough, bake dough
    - abstracted version: bake bread

Little/No Delegation	Some Delegation	Much Delegation
decide on grain type	decide on grain type	decide on grain type
buy the appropriate seed type; learn about growing techniques		
grow grain	locate and buy grain from farmer • packed in different sizes	
harvest grain		
bring grain inside to grinding room	transport grain to grinding place	
buy grinder (if needed) load hopper of grinder	get grain milled • the mill may stipulate input conditions, e.g., min size of packaging	place and receive order: "I'd like a loaf of <fill in the blank>"
grind grain (repeat until enough grain obtained)		
secure yeast, water	secure yeast, water	
prepare dough	prepare dough	
bake bread	bake bread	
let bread cool and slice	let bread cool and slice	
eat bread	eat bread	eat bread

No Delegation of Task…

```
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);
   }
}
```

- computes the area of a rectangle.
- code handles both storage (of data) and computation (of area).
- computation is a simple expression.

No Delegation of Task…

$$\text{BMI} = \left(\frac{\text{Weight in Pounds}}{(\text{Height in inches}) \times (\text{Height in inches})} \right) \times 703$$

or

$$\text{BMI} = \frac{\text{Weight in Kilograms}}{(\text{Height in Meters}) \times (\text{Height in Meters})}$$

we want to compute the Body Mass Index (BMI) for an particular individual

```
double weightInLbs = 170.0;
int heightInInches = 5*12+9; // this is the height 5'9";
double bmi = weightInLbs
           / (heightInInches*heightInInches) * 703;
```

PS:
what if
`weightInLbs`
were an int ???

- code handles both storage (of data) and computation (of BMI).
- computation is somewhat straightforward.

Delegation of Task… (§2.1.1)

```
double weight = 170.0;
String height = "5'9";
double bmi = ToolBox.getBMI(weight, height);
```

- Data storage
  - no delegation
  - "We" (the main method) take care of data storage by declaring ints
- Computation
  - delegation to a **static** *method* within a class.
- Any method must be one of the following:
  - static
  - non-static

Delegation of Task+Storage… (§2.1.2)

```
// 8.5 inches is approx 22 cm
// 11 inches is approx 28 cm
Rectangle letterSizedPaper = new Rectangle(22, 28);
double area = letterSizedPaper.getArea();
```

- Data storage
  - delegation to object of *type* `Rectangle`
  - we make use the object by using its reference **`letterSizedPaper`**
  - We can access width and height by
    - **`letterSizedPaper.getWidth()`** and **`letterSizedPaper.getHeight()`**
- Computation
  - delegation to a *method* within the class.

About methods…

- A method must belong to a class.
  - methods cannot exist in any other fashion
- Methods perform tasks and are named accordingly:
  - actions or verbs
    - e.g., `computeBMI(double, String)`
  - complete predicate
    - e.g., `isEnabled()`
- Methods have returns:
  - `void` or a data type

About method invocation…

- A method invocation **must** be followed by its parameters
  - a pair of parenthesis with zero or more parameters sandwiched in between
  - e.g.,
    - **`ToolBox.getBMI(weight, height);`**
    - **`output.println("Hello");`**

About method invocation…

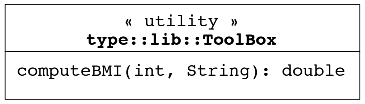
- Classes provide services to clients.
  - *methods* are one category of service
  - *fields* are another category of service
- Clients (you) ***must indicate the source of the method***: [one of the following]
  - ClassName.method(…) [this is for static methods]
    - e.g.,
      - `ToolBox.getBMI(weight, height);`
  - variable.method(…) [this is for non-static methods]
    - e.g.,
      - `output.println("Hello");`
      - `letterSizedPaper.getArea();`

What is "signature" ?

- the signature of a method is
  - the method name **together with**
  - the types of its parameters
  - e.g.,
    - **`computeBMI(double, String)`**
    - **`println(String)`**
  - The method's **return** is not considered to be part of the method's signature
- **The methods in a class must be unique**

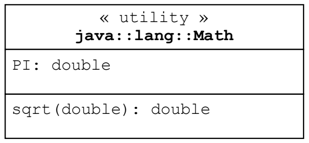
UML (Unified Modeling Language)

The class diagram of a utility class in the TYPE library:



```
« utility »
type::lib::ToolBox
computeBMI(int, String): double
```

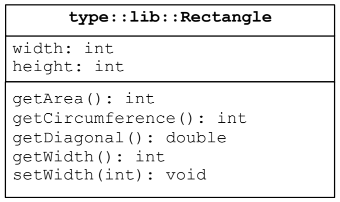
The class diagram of a utility class in the Java library:



```
« utility »
java::lang::Math
PI: double
sqrt(double): double
```

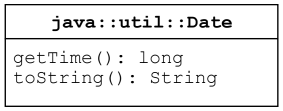
UML (of a Non-Utility)

A class diagram from the TYPE library:



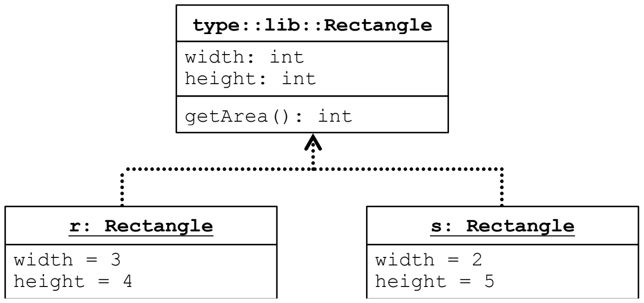
```
type::lib::Rectangle
width: int
height: int
getArea(): int
getCircumference(): int
getDiagonal(): double
getWidth(): int
setWidth(int): void
```

A class diagram from the Java standard library



```
java::util::Date
getTime(): long
toString(): String
```

UML: An Object Diagram



Take Home Points

- Do you know
  - the difference between <u>an object reference</u> and an <u>object</u>?
  - how to recognize the use of a static method?
  - how to recognize the use of a non-static method?
  - how to declare an object reference?
  - how to assign the object reference to refer to a particular object?
  - how to use a static method?
  - how to use a non-static method?