CSE 1710

Lecture 5
What is Delegation?

The assigned reading was:

- **What is Delegation?**
  - **sec 2.1, pp. 48–58**
  - Key Concepts 2.1–2.9 [2.10]

Key Concept 2.1

Today's applications must **delegate** some or all of its work to other **classes**.

Key Concept 2.2

In the **procedural** paradigm, the program invokes methods **on** the class.

Key Concept 2.3

A **method** has a **signature** (name and parameter types) and a **return** type (type of the returned value). If a method is **void**, then it has no return. Data is **passed** to a method through **parameters**.

Key Concept 2.4

In the **modular** paradigm, the program accesses attributes and invokes methods **on** the class.

An **attribute** has a name and a type and allows data to **persist**.

Attributes and methods are known collectively as **features**.

Key Concept 2.5

In the **object-oriented** paradigm, the program instantiates the class and uses the created object (rather than the class).

This paradigm is also known as **OOP** (for object oriented paradigm).

Whereas a **class** has only attributes and methods, an **object** also has an object reference and a state (the value of all its attributes).

Key Concept 2.6

A **class diagram** in UML (Unified Modelling Language) represents a class definition.

In such diagrams, a class is depicted as a rectangle with one or more compartments.

The top compartment is mandatory and contains the class name (possibly quantified) and an optional **stereotype**.

The next two compartments are optional: one for attributes and one for methods.

Key Concept 2.7

A utility class cannot be instantiated.

For a utility class, all features are said to be static.

All features are accessed/invoked **on the class name**.

Such classes are denoted in UML (Unified Modelling Language) by the stereotype `<<utility>>`

Key Concept 2.8

An **object diagram** in UML represents an object.

In such diagrams, an object is depicted as a rectangle with two compartments: the object's identity is in the top compartment and its state in the bottom one.

Key Concept 2.9

In UML, a dashed line can be placed between:

- a class diagram and a class diagram
- a class diagram and an object diagram

Between class diagram A and B means **class A delegates to class B**

Between class diagram A and object diagram X means the **object X is an instance of class A.**

Key Concept 2.10

Abstraction is a strategy for replacing complexity with simplicity. **Layered abstractions** give rise to **abstraction** hierarchies in which **higher**-level abstractions have fewer details.

Tasks you should be able to perform:

- Recognize the delegation of a task and the delegation of representation within an application
- Explain the difference between an object reference and an object

Tasks you should be able to perform:

- Recognize the use of a static method
- Recognize the use of a non-static method
- How to declare an variable to represents an object reference
- How to obtain an object reference and to store the reference for subsequent use
- How to use a static method
- How to use a non-static method

Little/No Delegation
decide on grain type …*rye*
buy the appropriate seed type; learn about growing techniques
grow grain
harvest grain
bring grain inside to grinding room
buy grinder load hopper of grinder
grind grain (repeat until enough grain obtained)
secure yeast, water
prepare dough
bake bread
let bread cool and slice
eat bread

Little/No Delegation	Some Delegation
decide on grain type …*rye*	decide on grain type …*rye*
buy the appropriate seed type; learn about growing techniques	
grow grain	**place an order for a bag of grain** • need to find the farmer • need to figure out what size of bag to buy (see next step)
harvest grain	
bring grain inside to grinding room	transport grain to grinding place
buy grinder load hopper of grinder	**place an order to get grain milled** • need to find a mill • need to ensure that you give them the correctly-sized bag (e.g., the mill may stipulate input conditions, such as min size of packaging)
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

Little/No Delegation	Some Delegation	Much Delegation
decide on grain type …*rye*	decide on grain type …*rye*	decide on grain type …*rye*
buy the appropriate seed type; learn about growing techniques		
grow grain	**place an order for a bag of grain** • need to find the farmer • need to figure out what size of bag to buy (see next step)	
harvest grain		
bring grain inside to grinding room	transport grain to grinding place	
buy grinder load hopper of grinder	**place an order to get grain milled** • need to find a mill • need to ensure that you give them the correctly-sized bag (e.g., the mill may stipulate input conditions, such as min size of packaging)	place an order: "I'd like a loaf of sliced *rye* bread"
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

Example 1: Body Mass Index

Body Mass Index (BMI) is a heuristic for estimating in individual's body fat based on that individual's height and weight.

It is inexact (e.g., it *overestimates* body fat for athletes and *underestimates* body fat for those with low lean body mass.

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

or

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

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

weight: 170 pounds
height: 5'9"

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

YUCK! "magic number"
We'll leave it for now…

these statements handle both storage (of data) and computation (of BMI).

the computation is somewhat straightforward.

what bad thing would happen if `weightInLbs` were to be declared as `int`?

But let's look at the class called `ToolBox`

http://www.cse.yorku.ca/java/api/type/api/

```
public static double getBMI(double weight,
                            java.lang.String height)
```

Compute the body mass index.

Parameters:
`weight` - the weight in pounds. To be valid, weight must be positive.
`height` - the height in feet'inches. To be valid, height must have a feet component (a positive integer) optionally followed by an inches component (a non-negative integer less than 12). And if both components are present then they must be separated by a single quote.
Returns:
the body mass index (BMI) for the given weight and height
Throws:
`java.lang.RuntimeException` - if either weight or height is not valid as defined above.

How nice!

There is a **service** that will compute BMI for us.
We can **delegate** this task to the class `ToolBox`

So how do we use this service?

FIRST QUESTION

what type of method is this? static or non-static

Is this a static method?

```
public static double getBMI(double weight,
                            java.lang.String height)
```

Compute the body mass index.

Parameters:
`weight` - the weight in pounds. To be valid, weight must be positive.
`height` - the height in feet'inches. To be valid, height must have a feet component (a positive integer) optionally followed by an inches component (a non-negative integer less than 12). And if both components are present then they must be separated by a single quote.

Returns:
the body mass index (BMI) for the given weight and height

Throws:
`java.lang.RuntimeException` - if either weight or height is not valid as defined above.

Is this a static method?

What value is returned?

What parameters do we need?

What is the contract?

```
public static double getBMI(double weight,
                            java.lang.String height)
```

Compute the body mass index.

Parameters:
`weight` - the weight in pounds. To be valid, weight must be positive.
`height` - the height in feet'inches. To be valid, height must have a feet component (a positive integer) optionally followed by an inches component (a non-negative integer less than 12). And if both components are present then they must be separated by a single quote.

Returns:
the body mass index (BMI) for the given weight and height

Throws:
`java.lang.RuntimeException` - if either weight or height is not valid as defined above.

```
double bmi;
double weight = 170.0;
String height = "5'9";
bmi = ???
```


What is the contract?

A paraphrase…

"Adhere to these specifications for the parameters and I promise to return to you the correct BMI value. If you do not adhere, I promise to throw an exception"

So to use the method…

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

We have delegated computation, but NOT storage.

Compare

```
double weightInLbs = 170.0;
int heightInInches = 5*12+9;
double bmi = weightInLbs
             / (heightInInches*heightInInches) * 703;
```

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


Example 2: Surface Area of Paper

Suppose we want to derive the surface area of various types of paper (letter size, legal size, A3, A4, etc)

Suppose we want to compute the surface area of a letter sized sheet of paper

width: 21.59 cm (8.5 inches)
height: 27.94 cm (11 inches)

```
double widthInCM = 21.59;
double heightInCM = 27.94;
double surfaceAreaLetterSizedPaper =
     widthInCM * heightInCM;
```

these statements handle both storage (of data) and computation (of surface area).

the computation is somewhat straightforward.

But let's look at the class called `Rectangle`

http://www.cse.yorku.ca/java/api/type/api/

type.lib
Class Rectangle

```
java.lang.Object
  └ type.lib.Rectangle
```

All Implemented Interfaces:
java.io.Serializable

```
public class Rectangle
extends java.lang.Object
implements java.io.Serializable
```

A class that encapsulates a rectangle.

Method Summary

boolean	`equals(java.lang.Object other)` Determine if this rectangle is the same as the passed one.
int	`getArea()` Determine the area of this rectangle.
int	`getCircumference()` Determine the circumference of this rectangle.
double	`getDiagonal()` Determine the length of the diagonal of this rectangle.
int	`getHeight()` Determine the height of this rectangle.
int	`getWidth()` Determine the width of this rectangle.
int	`hashCode()` Compute a hash code for this Rectangle.
void	`setHeight(int height)` Mutate the height this rectangle.
void	`setWidth(int width)` Mutate the width this rectangle.

How nice!

There is a **service** that represents rectangles for us.
It provides various operations **get area!**
We can **delegate** to the class `Rectangle`

So how do we use this service?

```
public int getArea()
```

Determine the area of this rectangle.

Returns:
the area of this rectangle

type.lib
Class Rectangle

```
java.lang.Object
  └ type.lib.Rectangle
```

All Implemented Interfaces:
java.io.Serializable

```
public class Rectangle
extends java.lang.Object
implements java.io.Serializable
```

A class that encapsulates a rectangle.

FIRST QUESTION

what type of method is this? static or non-static

What is "this" rectangle referring to exactly??

We need to use the services of `Rectangle` to first represent the shape and then to perform computation….

```
public Rectangle(int width,
                 int height)
```

Construct a rectangle with the passed width and height.

Parameters:
`width` - the width of the rectangle to construct.
`height` - the height of the rectangle to construct.

```
//double widthInCM = 21.59;
int widthInCM = 22; // the constructor needs ints!!!
//double heightInCM = 27.94;
int heightInCM = 28; // the constructor needs ints!!!

Rectangle letterSizedPaper;
letterSizedPaper = new Rectangle(widthInCM, heightInCM);
```


Is this a static method?

What value is returned?

What parameters do we need?

```
public int getArea()
```

Determine the area of this rectangle.

Returns:
the area of this rectangle

```
//double widthInCM = 21.59;
int widthInCM = 22;
//double heightInCM = 27.94;
int heightInCM = 28;

Rectangle letterSizedPaper;
letterSizedPaper = new Rectangle(widthInCM, heightInCM);

double surfaceAreaLetterSizedPaper;
surfaceAreaLetterSizedPaper = letterSizedPaper.getArea()
```

We have delegated computation, AND storage.

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…

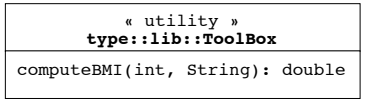
- Classes provide services to clients.
  - *methods* are one category of service
  - *fields* are another category of service
- Clients (you) ***must indicate the source of the method***: [one of the following]
  - ClassName.method(…) [this is for static methods]
    - e.g.,
      - `ToolBox.getBMI(weight,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

What is "signature" ?

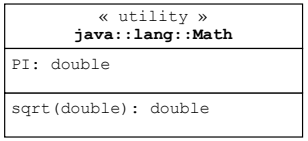
- the signature of a method is
  - the method name **together with**
  - the types of its parameters
  - e.g.,
    - `computeBMI(double, String)`
    - `println(String)`
  - The method's **return** is not considered to be part of the method's signature
- **The methods in a class must be unique**

UML (Unified Modeling Language)

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

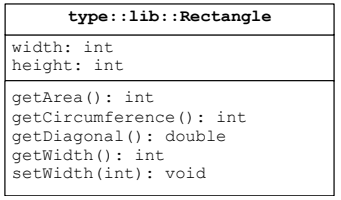


The class diagram of a utility class in the Java library:

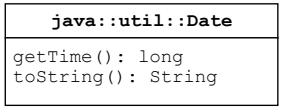


UML (of a Non-Utility)

A class diagram from the TYPE library:



A class diagram from the Java standard library



UML: An Object Diagram