Unit 02

Basics of Object Oriented Programming

CMPS 251, Fall 2020, Dr. Abdulaziz Al-Ali

Checkpoint

- What is new about switch statements in Java?
- Which class did we use to read inputs?
- What is wrong with this code (hint: two mistakes)?

```
public void PrintMyBalance(int x)
{       System.out.println("My balance is:" + x);  }

PrintMyBalance(44.2);
```

Objectives

- Introduce the concept of object oriented programming
- Describe OOP concepts such as objects, variables, methods, constructors, getters, and setters

Intro to OOP

- Object oriented programming involves designing your program around the data it operates on
- Objects are designed to store data and operate on it

Procedural Example

- Write a simple program to calculate the area of a rectangle that is 5cm by 7cm
- Standard, procedural solution that focuses on operations first:

```
public static void main(String[] args) {
    int width = 5;
    int height = 7;
    int area = width * height;

    System.out.printf("Area is %d\n",area);
}
```

Object Oriented Solution?

- Think about the data
- Describe a rectangle
  - What properties does it have? (Attributes)
  - What operations do we want to perform with it? (Methods)

- Objects have two general capabilities:
  - Store data in **attributes/fields**
  - Perform operations using **methods**

Object Oriented Solution

- First, define what a rectangle looks like:

```
public class Rectangle {
    public int width;
    public int height;

    public int getArea() {
            return width*height;
    }
}
```

- Next, use it to solve the problem:

```
public class App {
    public static void main(String[] args) {
            Rectangle r = new Rectangle();
            r.width = 5;
            r.height = 7;
            System.out.printf("Area is %d\n", r.getArea());
    }
}
```

Some Terms

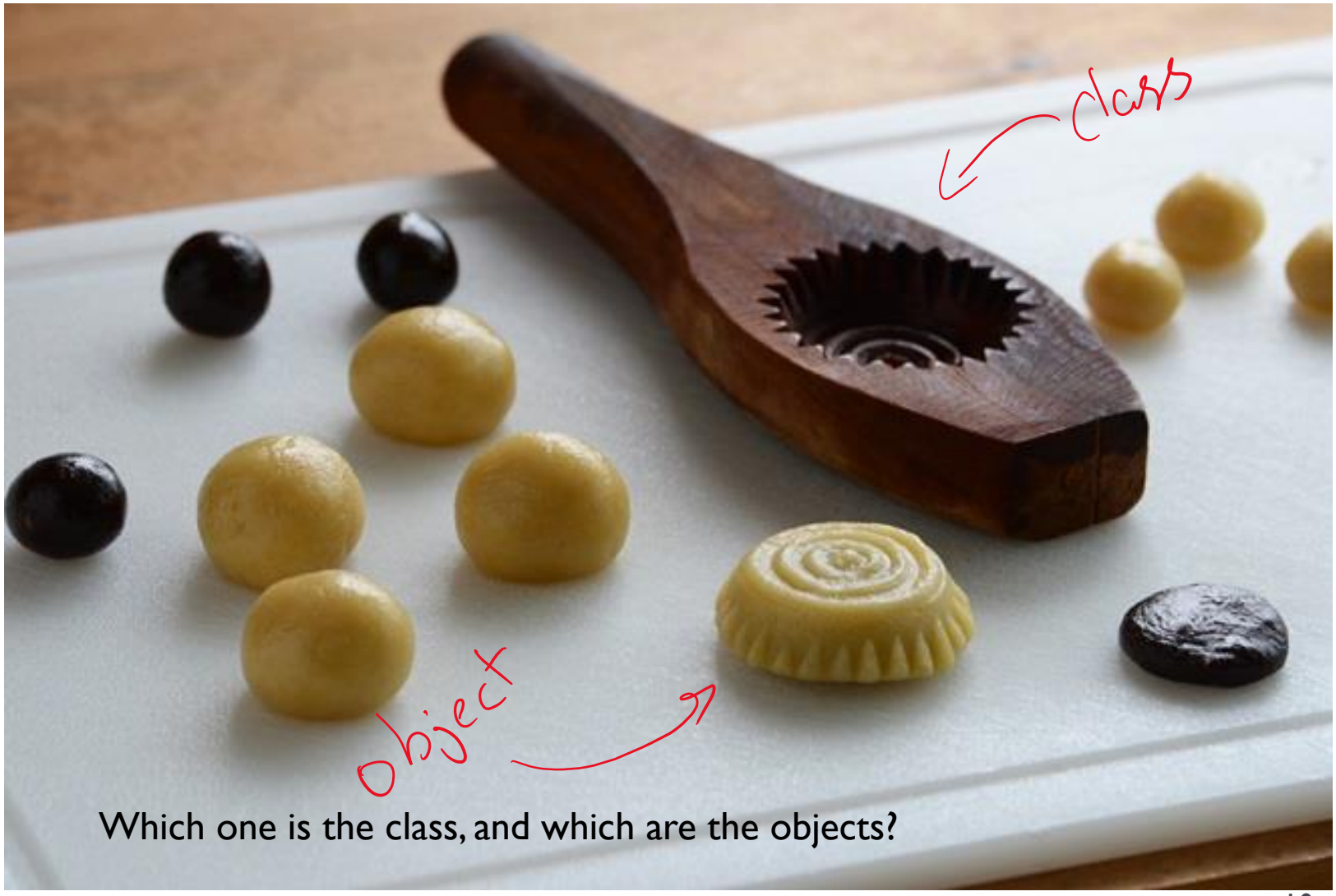
- A **Class** defines the attributes and methods

```
public class Rectangle {
   public int width;
   …
}
```

- An **Object** is an instantiation of the class.

```
Rectangle r = new Rectangle();
```

Cookies?



Which one is the class, and which are the objects?

Classes in Java

- Classes: Where Objects Come From
  - A *class* is code that describes a particular type of object. It specifies the data that an object can hold (the object's **fields**), and the actions that an object can perform (the object's **methods**).
  - A class is a code, a blue print from which individual objects are created.
  - A class can contain **variables**, **constructors** and **methods**.

Last Lecture

- Classes Vs. Objects
  - Difference? (Very important)

- What are the components of a class?

Demo

- See Bank package/example in the code sample of this unit.
  - Do TODOs 1-10

Constructors

- Objects should have constructors that allow you to define the value of attributes at object creation

```
public class Rectangle {
    public int width;
    public int height;

    public Rectangle(int w, int h) {
        this.width = w;
        this.height = h;
    }

    public int getArea() {
        return width*height;
    }
}
```

```
public class App {
    public static void main(String[] args) {
        Rectangle r = new Rectangle(5,7);
        System.out.printf("Area is %d\n",
                r.getArea());
    }
}
```

The **this** Variable

- The `this` object reference can be used inside any non-static method to **refer to the current object**
  - `this` is used to **refer to the object** from inside its class definition
  - The keyword `this` stands for the receiving object
- `this` is commonly used to **resolve name conflicts**
  - Using **`this`** permits the use of attributes in methods that have local variables / parameters with the same name

```
public void setName(String name) {
        this.name = name;
}
```

This is a parameter

This is an attribute. To avoid confusion we add **this.** in-front of it

Philosophical questions

- When you are designing a class, who are you designing it for?

- Why does it make sense to use classes?

Last Lecture

- Constructors:
  - What do they do?
  - Can we have more than one in the same class? Why is that helpful?
  - When do we put the word "this" in front of instance variable names?

Constructors

- Called with the **new** keyword when creating the object
- Every class has a constructor. A default (empty) constructor is created automatically *only if you do not explicitly create one*
- A class may have more than one constructor. At least one of them will be called when object is created.
- Constructors usually set initial values of fields and initial work setup

Demo

- See the *bank* example again in the sample code of this unit (TODOs 11 and 12)

A note about constructors

- Can a constructor be re-used by another constructor?!

A note about constructors

- Can a constructor be re-used by another constructor?!

- Yes. What are the rules?
  - Must be called first in the constructor body using the word "this".

Last Lecture

- Constructors.

- What is the magic method we used to teach Java how to represent an object in String format?

Demo

- Lets do the TODO elements inside the package constructors: see classes Rectangle and Person.

Access Modifiers

- To use a modifier, you include its keyword in the definition of a class, method, or variable. The modifier precedes the rest of the statement.

- Java provides a number of access modifiers to set access levels for classes, variables, methods and constructors. The four access levels are:
  - Visible to the world (**public**).
  - Visible to the package and all subclasses (**protected**).
  - Visible to the package. The default. No modifiers are needed.
  - Visible to the class only (**private**).

Access Modifiers

- Access levels:

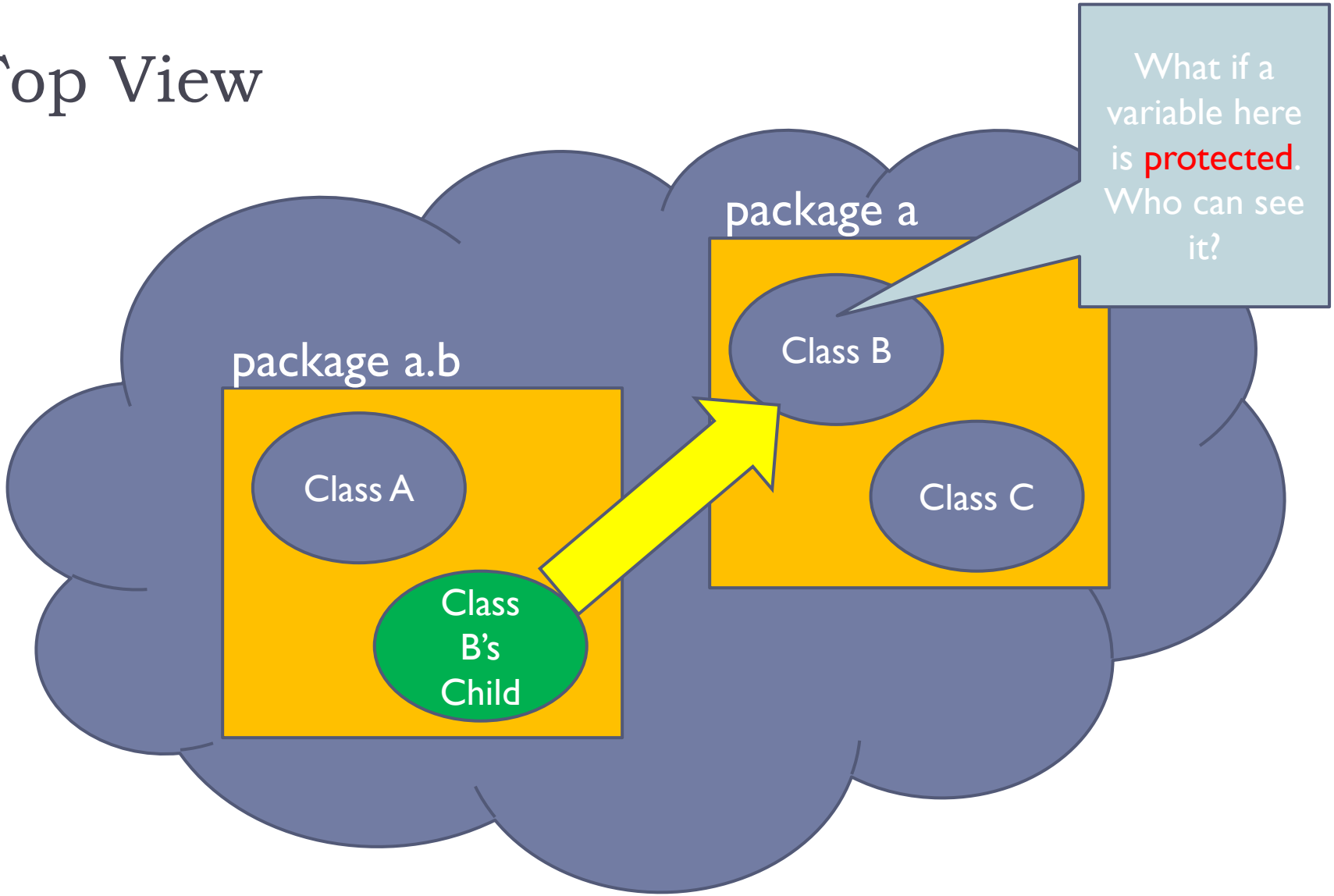
	Class	Sub-classes	Package	Everyone!
public				
protected				
(nothing)/package access				
private				

Access Modifiers

- Access levels:

	Class	Sub-classes	Package	Everyone!
public	✓	✓	✓	✓
protected	✓	✓	✓	
(nothing)/package access	✓		✓	
private	✓			

Top View


What if a variable here is protected. Who can see it?
package a
Class B
Class C
package a.b
Class A
Class B's Child

Non-Access Modifiers

- The ***static*** modifier for creating class/static methods and variables
- The ***final*** modifier for finalizing the implementations of classes, methods, and variables.
- The ***abstract*** modifier for creating abstract classes and methods.
- The ***synchronized*** and ***volatile*** modifiers, which are used for threads.

Demo

- See the *apples* package in the code sample of this unit (TODO 1-3).

Last lecture

- Which access modifier is more restrictive: package/default, or protected?
- What are none-access modifiers?
- What does the *final* modifier mean?
- What does the *synchronized* modifier mean? (extra)

Variables

- **Local variables:** variables defined inside methods, constructors or blocks are called local variables. The variable will be declared and initialized within the method and the variable will be destroyed when the method has completed.

- **Instance variables:** Instance variables are variables defined inside a class but outside any method. These variables are instantiated when objects are created. Instance variables can be accessed from inside any method, constructor or blocks of that particular class.

- **Static (or, Class) variables:** Class variables are variables declared within a class, outside any method, with the static keyword.

Static Attributes (variables)

- If a variable in a class is defined as static, then there is only **one copy** of it belonging to the class
  - All objects of that class share that same one copy
  - Any change in the static variable can be seen by all objects

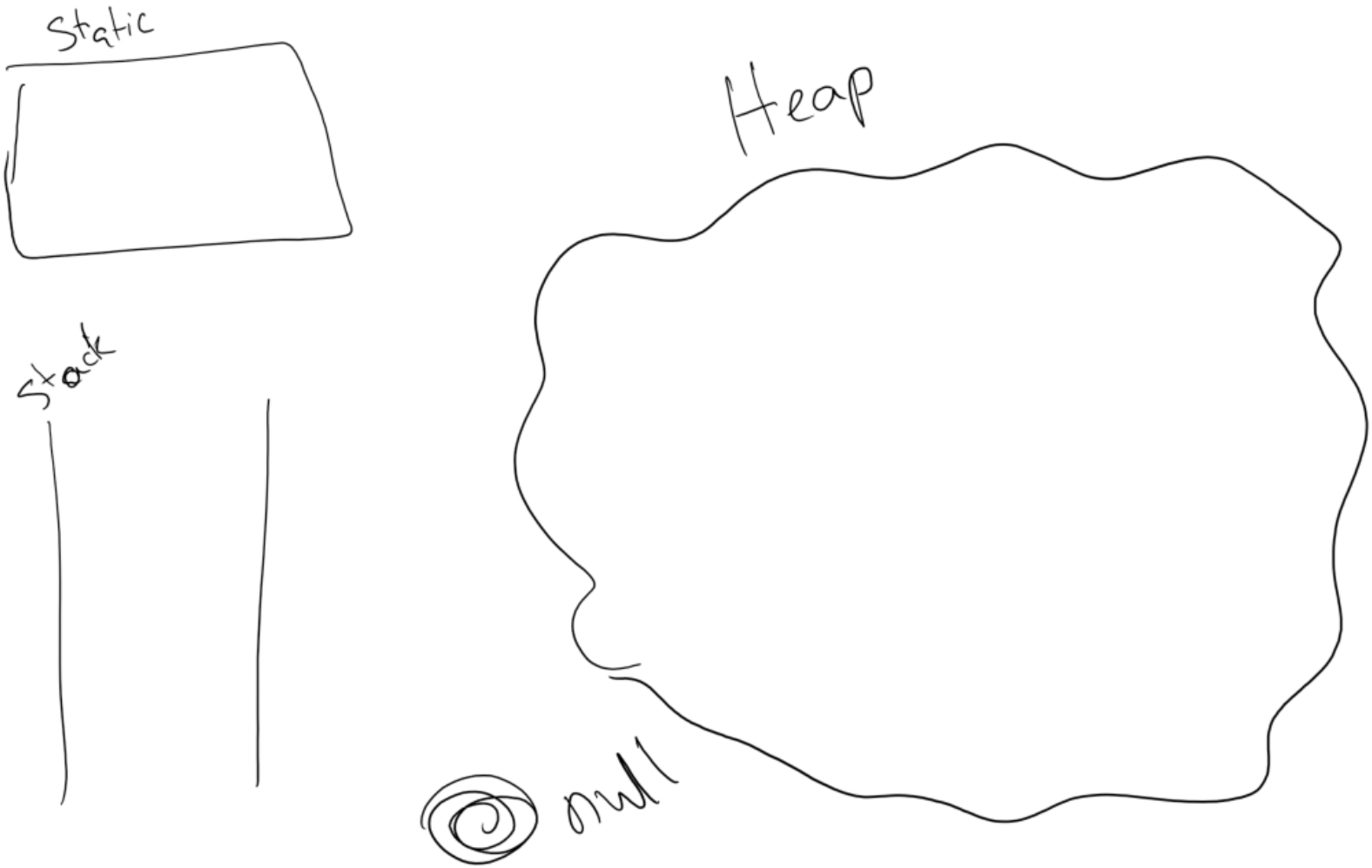
Last lecture

- Memory simulation:
  - What are the three types of memory we have seen last lecture?
  - Which of them is the largest?
  - What is special about static variables?
  - Where do objects go?
  - What is special about the stack memory?

Demo

- Simulate the program in the *memorytypes* package in the code sample of this unit.

Memory Simulation Demo



Try to simulate the classes inside the *memorytypes* package yourself.

Local variables

- Declaration location — Inside methods, and constructors
- Creation/destruction — Function entry → exit
- Access modifiers? — Not needed
- Visibility? — Inside methods, and constructors
- Heap or stack?
- Default values: none

Instance variables

- Declaration location — Inside classes, but outside of methods
- Creation/destruction — When a **new** object is created → Garbage Collector
- Access modifiers? — Yes
- Visibility? — At least in the class itself
- Heap/stack? — Heap
- Default values: false, 0, null

Static variables

- Declaration location — Inside classes, but outside of methods
- One copy !
- Creation/destruction — Application Start → End
- Access modifiers? — Yes, but usually public
- Visibility? — At least within the class
- Heap/stack? — Static!
- Default values: false, 0, null

Reminder

Quiz 1 end of next lecture Thursday (Sep 10th)

- **Content:**

  Units 1 and 2 (everything before arrays)

- **Time:**

  20 minutes (end of lecture on Thursday 10/9)

- **Format:**
  - Online blackboard, using Lockdown browser
  - Multiple choice, true/false, item matching

Make sure you test your laptop on the sample Quiz. See "Lockdown browser" announcement

Last Lecture

- Local variables
  - Where are they defined?
  - Their life cycle
- Static variables
  - How many copies in memory?
- Instance variables
  - When do they get created? And destroyed?

Methods

- A Method is used to implement a behavior of the class and is accessible via an object of the class.
- A method has an **access modifier**, **return type**, **name**, and optionally a **set of parameters**.
- A common category of methods is the set and get methods to access *private* attributes of the class.
- A class can have any number of methods.
- Methods that are **static** can be accessed by the class name without the need to create an object from the class. Such methods are common in ***utility*** classes such as the **Math** class in Java..

Example

Access Modifier

Return Type

Name

```
public int getArea() {
        int area = width * height;
        return area;
}
```

Parameter

```
public void setWidth(int width) {
    this.width = width;
}
```

Static Methods

- If a method in a class is declared static, then it is called <u>directly from the class</u> (and not an object).
- Static methods cannot access non-static fields or methods from a class.

Why is that?

- Example: Math package in Java
  - `Math.sqrt(40);`
- The Math package has lots of static methods you can call *without* creating a Math object
  - See them here: https://docs.oracle.com/javase/8/docs/api/java/lang/Math.html

What happens if…

- We try to use *instance* variables from a static method?

Demo

- See *TODO 13-16* in the *bank* package in the sample code of this unit.

Getters and Setters

- Typically you declare most attributes of a class private and provide methods to get and set their values
- The methods that retrieve the data of attributes are called **getters** or **accessors**
- The methods that modify the data of attributes are called **setters** or **mutators**

Example

Instance variable

Static variable

Local variable

```
public class Rectangle {
    private int width;
    private int height;
    public static int MX_HEIGHT = 15;

    public Rectangle(int w, int h) {
        this.width = w;
        this.height = h;
    }

    public int getWidth() {
        return width;
    }

    public void setWidth(int width) {
        this.width = width;
    }

    public int getHeight() {
        return height;
    }

    public void setHeight(int height) {
        this.height = height;
    }

    public int getArea() {
        int area = width * height;
        return area;
    }
}
```

Attributes

Constructor

Getter

Setter

Getter

Setter

Method

Summary

- An object stores data and operations on that data
- Objects have attributes and methods