香港浸會大學
HONG KONG BAPTIST UNIVERSITY

COMP2026

Problem Solving Using Object Oriented Programming

OOP Part 2

Overview

- The equals method
- Method signatures & Method Overloading
- Default Constructor & Copy Constructor
- Multi-class Java Applications
- public vs. private
- The `static` keyword

Remarks on a few Special Methods

A few special methods typically in a class...

- `toString` -
  Converts this object into a `String`; good for debugging
- `equals` - Compares this object with another object (`true` if identical; `false` otherwise)
- **Constructor** - Constructs and prepare the object before use

The Point Class

Let's begin our discussion using the Point Class

```
public class Point {
    private final int x;
    private final int y;

    public Point(int x, int y) {
        this.x = x;
        this.y = y;
    }
}
```

- `final` – a keyword, indicating that the variable cannot be updated after its initialization
- Task: Add `getter` methods and `toString` to `Point`
- Next, add an `equals` method to `Point`

The equals Method #1

Let's add the equals method to Point

```
public boolean equals(int x, int y) {
    return this.x == x && this.y == y;
}
```

- The `equals` method #1 accepts two integers, `x` and `y`, as parameters
- It check whether the `x` and `y` coordinates of the current Point object are the same as the given ones
- Next, add one more `equals` method to `Point`

The equals Method #2

Let's add one more equals method to Point

```
public boolean equals(Point p) {
    return this.x == p.x && this.y == p.y;
}
```

- The equals method #2 accepts a `Point` object, `p`, as parameter
- It check whether the `x` and `y` coordinates of the current Point object are the same as those of `p`'s

Method Overloading

- Note that we now have two equals methods. Which one do we use??
- That depends on the arguments provided

```
//equals Method #1
public boolean equals(int x, int y) {
    return this.x == x && this.y == y;
}

//equals Method #2
public boolean equals(Point p) {
    return this.x == p.x && this.y == p.y;
}
```

Method Overloading


```java
//equals Method #1
public boolean equals(int x, int y) {
    return this.x == x && this.y == y;
}

//equals Method #2
public boolean equals(Point p) {
    return this.x == p.x && this.y == p.y;
}
```

- Calling which constructor depends on the arguments you provide
- If you do `p1.equals(1, 2)`, method #1 is called
- If you do `p1.equals(p2)`, method #2 is called
- This is called **method overloading**
- Note: `p1` & `p2` are Point objects

Method Overloading

- In Java, we can have many methods of the same name as long as the data types of their parameter lists are different
- **Method name** – is just a method name
- **Method signature** – is the method name plus the data type of the parameters (but not the names of the parameters)
- For example:
  - equals method #1: equals-int-int
  - equals method #2: equals-Point
- This is called **method overloading** as we are overloading the method name
- Overloading methods may have **different return types**

Method Overloading

- Note that we only keep the data type of the parameters, but not name of the parameters
- To the Java compiler, it will see the following as the same:

1.

```
public boolean equals(int x, int y) {...}
```

Signature: equals-int-int 2.

```
public boolean equals(int y, int x) {...}
```

Signature: equals-int-int

- Same method signature ➡️ they are the same ➡️ not acceptable by Java

More on the Two equals Methods

Improving equals method #2

```
public boolean equals(Point p) {
    return equals(p.x, p.y));
}
```

- Line 2 of the `equals` method #2 could be improved to the one shown above
- It calls the `equals` method #1 and returns the result (Improved?? Why???)
- In case we want to revise how we check for equality, we only need to update the original one
- Also, if the original one works, this one works too!

Notes on the equals Method

- When developing a new class, developers are recommended to provide the `equals` method (also the `toString` method)
- Typically, only the developer of a class can decide how to check for the *equality of two objects*
- Best if the developer of a class to provide methods for testing equality

`equals` vs `compareTo`

In the String class

- `str1.equals(str2)` [returns `boolean`]
  - `true` if str1 and str2 are identical
  - `false` if str1 and str2 are not identical
- `str1.compareTo(str2)` [returns `int`]
  - 0 if str1 and str2 are identical
  - -ve – if str2 is lexicographically less than str1 (e.g., str1 is "apple" and str2 is "orange")
  - +ve – if str2 is lexicographically greater than str1 (e.g., str1 is "dog" and str2 is "cat")

Default Constructor

Default Constructor for the Point Class

```
public Point() {
    this.x = 0;
    this.y = 0;
}
```

- This constructor takes no parameters
- This is called the **default constructor**
- This is invoked by `new Point()`

Default Constructor

Default Constructor for the Point Class

```
public Point() {
    this(0, 0);
}
```

- With similar argument as the two equals methods, the default constructor presented on the last slide could be improved to the one shown above
- Line 2 of the above, `this(0, 0);` will call the original constructor with "(0, 0)"
- In case we want to update the constructors, we only need to update the original one
- To call a constructor of a class from another constructor of the same class, you can use the `this` keyword followed by parentheses, "()", containing the constructor arguments

Copy Constructor

香港浸會大學
HONG KONG BAPTIST UNIVERSITY

Copy Constructor for the Point Class

```
public Point(Point p) {
    this(p.x, p.y);
}
```

- This constructor constructs a new object based on another existing object of the same class
- This is called the **Copy Constructor**
- In total, we now have three constructors (the original one, and the two new ones). Which one do we use??
- Method overloading ➡️ depends on the arguments provided

Which Constructor???

香港浸會大學
HONG KONG BAPTIST UNIVERSITY

The Original Constructor

```
public Point(int x, int y) {
    this.x = x;
    this.y = y;
}
```

The Default Constructor

```
public Point() {
    this(0, 0);
}
```

The Copy Constructor

```
public Point(Point p) {
    this(p.x, p.y);
}
```

- Method Overloading – the arguments decides which constructor to call
- `new Point(1, 2)` Original constructor
- `new Point()` Default constructor
- `new Point(p)` Copy constructor

More on Constructors...

- To call the constructor of a class from another constructor of the same class, you can use the `this` keyword followed by parentheses, "()", containing the constructor arguments
- Such a constructor call must appear as the first statement in the constructor's body
- If no constructors are provided in a class, the compiler creates a default constructor
- If a class declares constructors, the compiler will not create a default constructor. In this case, you must declare a no- argument constructor if default initialization is required
- Next, create another class, `MyApp`, to work with `Point`

The MyApp Class


```
public class MyApp {
    public static void main(String [] args) {
        Point p1 = new Point(1, 1);
    }
}
```

- A class can create objects of another class
- In MyApp, try printing `p1.x`. Can you do it?
- In Point, try changing `x` from `private` to `public`, and try running `MyApp` again. Can you do it now?
- In Point, change `x` back to `private`

Public vs. Private


```
public class MyApp {
    public static void main(String [] args) {
        Point p1 = new Point(1, 1);
    }
}
```

- public – you allow other classes to access that member
- private – you do not allow other classes to access that member
- Yet, objects of the same class can always access those members

The Line Class

To Deepen our Understanding, the Line Class

```
public class Line {
    private final Point p1
    private final Point p2;

    public Line(Point p1, Point p2) {
        this.p1 = p1;
        this.p2 = p2;
    }
}
```

- Assume `p1` does not equal to `p2` (What if they are?? Handle later!)
- Task #1: Add the following Points to MyApp:

```
p2: ( 5, 5 ), p3: ( 2, 6 ), p4: ( 4, 8 )
```

- Task #2: Add a new Line to MyApp using p1 and p2

The Line Class


```
public class Line {
    private final Point p1
    private final Point p2;

    public Line(Point p1, Point p2) {
        this.p1 = p1;
        this.p2 = p2;
    }
}
```

- Try This: Can we add a new Line to MyApp using new Line()? What's wrong??
- The Line class does not have a default constructor
- Now, remove this new Line().

The Line Class


```
public class Line {
    private final Point p1
    private final Point p2;

    public Line(Point p1, Point p2) {
        this.p1 = p1;
        this.p2 = p2;
    }
}
```

- Task 1: Add a getter methods for p1 and p2 to Line (Trivial!)
- Task 2: Add toString method to Line (Trivial!)
- Task 3: Add a copy constructor to Line (Trivial!)
- Task 4: Add getSlope method to Line

The Line Class


```
public class Line {
    private final Point p1;
    private final Point p2;

    public Line(Point p1, Point p2) {
        this.p1 = p1;
        this.p2 = p2;
    }
}
```

- Task 5: Add `isParallelWith` method to Line
- Task 6: Add `isOnLine` method to Line
- Task 7: Add `equals` method to Line
- Task 8: Add a few more tests to the MyApp class to test our work

Task 4 of the Line Class

Adding the getSlope Method

```
public double getSlope() {
    double x1 = p1.getX();
    double y1 = p1.getY();
    double x2 = p2.getX();
    double y2 = p2.getY();
    return (y2-y1) / (x2-x1);
}
```

- The slope of a line can be calculated as:

$$\frac{y2-y1}{x2-x1}$$

- For accuracy reason, let's use `double`

Task 5 of the Line Class

香港浸會大學
HONG KONG BAPTIST UNIVERSITY

Adding the isParallelWith Method

```
private final double THRESHOLD = 0.001;


public boolean isParallelWith(Line line) {
    return Math.abs(this.getSlope() - line.getSlope()) < THRESHOLD;
}
```

- To check if two lines are parallel, check their slopes (same slope ➡️ parallel)
- `Math.abs` – calculates absolute value, a method from `Math` class
- To check if two floating point numbers are equal, check if their difference is smaller than `THRESHOLD` (a constant we defined)
- By convention, constants are indicated with all upper case

Task 6 of the Line Class

香港浸會大學
HONG KONG BAPTIST UNIVERSITY

Adding the `isOnLine` Method

```
public boolean isOnLine(Point p) {
    if (this.p1.equals(p) || this.p2.equals(p)) {
        return true;
    }
    Line line1 = new Line(this.p1, p);
    Line line2 = new Line(this.p2, p);

    return line1.isParallelWith(line2);
}
```

A point, p, is on the line, if...

1. p is the same as $p1$ or $p2$, or
2. the two lines, $\overline{pp1}$ and $\overline{pp1}$, are parallel

Task 7 of the Line Class

香港浸會大學
HONG KONG BAPTIST UNIVERSITY

```
public boolean equals(Line line) {
    return this.isOnLine(line.p1) && this.isOnLine(line.p2);
}
```

- A line `equals` to this line if p1 and p2 are both on this line

Points to Note about OO Design

Points to Note about Object Oriented Design:

- Methods are often short and do very simple and little thing
- A method often relies on the “service” provided by other methods
- Once a method is designed and developed well, tested thoroughly, it would be reliable and useable by others

```
public class Point {
    private static int ptCnt = 0;
    private final int serialNo;
    private final int x;
    private final int y;
```

- Now, modify our `Point` class by adding line 2 & 3
- And in the original constructor, add the following:

```
serialNo = ++ptCnt;
```

- Also, modify toString() as illustrated below:

```
return "Point-" + serialNo + ": (" + x + ", " + y + ")";
```

- Now, try running your program again

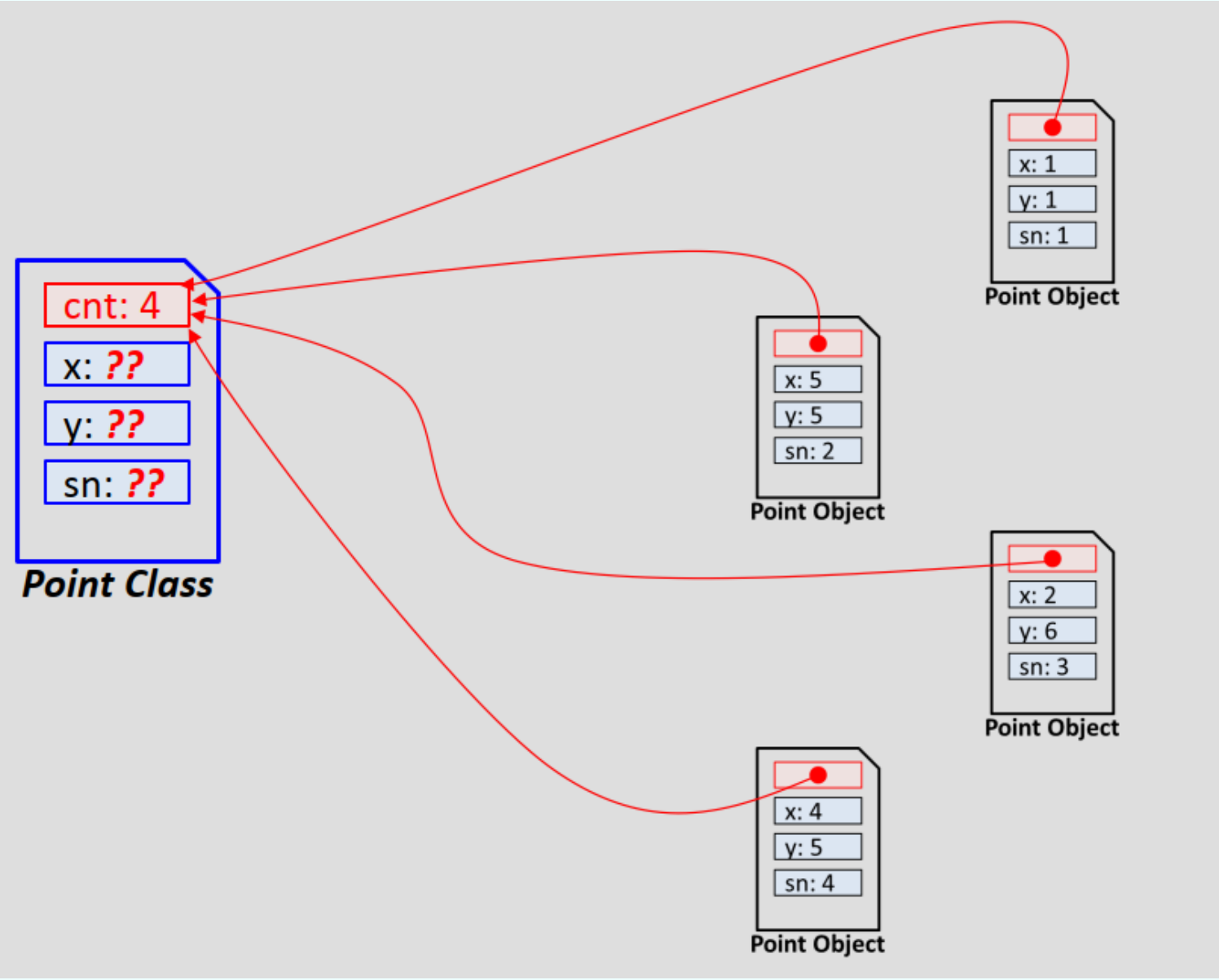
static

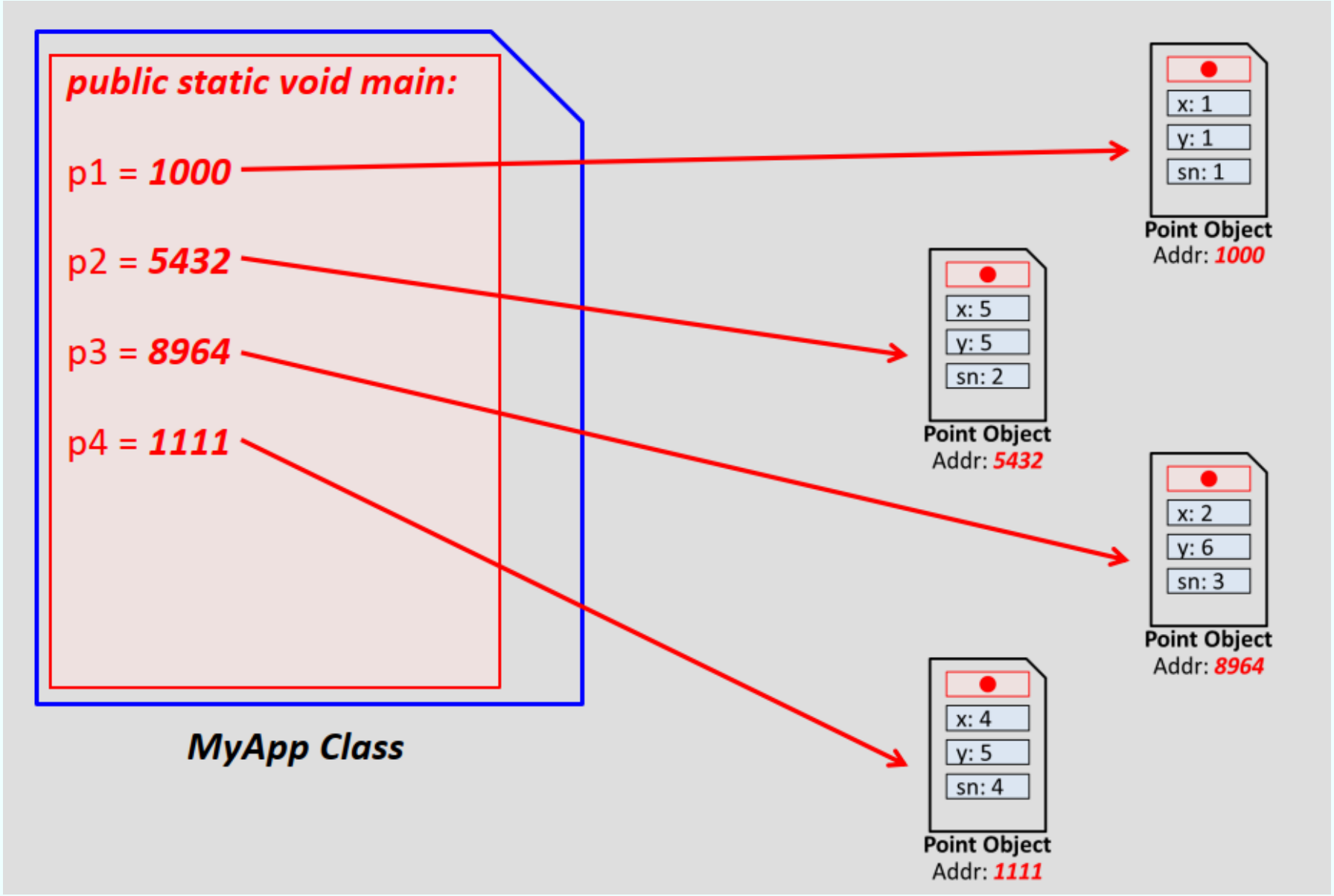
```
public static int getPtCnt() {
    return ptCnt;
}
```

- Try adding the above, and see what it returns
- `static` are members (fields or methods) shared among all objects of the same class
- `static` methods can be called without an object linking to it; should be called through the class

Examples:

- `public static void main(...)`
- `Integer.parseInt(str)`


cnt: 4
x: ??
y: ??
sn: ??
Point Class
x: 1
y: 1
sn: 1
Point Object
x: 5
y: 5
sn: 2
Point Object
x: 2
y: 6
sn: 3
Point Object
x: 4
y: 5
sn: 4
Point Object


public static void main:
p1 = 1000
p2 = 5432
p3 = 8964
p4 = 1111
MyApp Class
x: 1
y: 1
sn: 1
Point Object
Addr: 1000
x: 5
y: 5
sn: 2
Point Object
Addr: 5432
x: 2
y: 6
sn: 3
Point Object
Addr: 8964
x: 4
y: 5
sn: 4
Point Object
Addr: 1111

`static`

- A `static` variable represents **classwide** information that's shared among the class's objects
- Static variables have **class scope**, exist as soon as the class is loaded into memory
- Static members can be accessed by **any methods** of the same class (both static and non-static methods)
- A static method can access non-static members if an object reference is provided
- If no object reference is provided, a static method **cannot** access non-static members, because a static method can be called even when no objects of the class have been instantiated
- The `this` reference **cannot** be used in a static method

`static`

- A class's public static members can be accessed in two ways:

1. accessed by qualifying the member name with the class name (e.g., `Integer.parseInt(...)`)
2. accessed through a reference to any object of the class (e.g., `myInt.parseInt(...)` where `myInt` is an `Integer` object)

- Private static class members can only be accessed through methods provided by the class

More about `this` ...

- A non-static method of an object implicitly uses keyword `this` to refer to the object's instance variables and other methods
- If needed, the keyword `this` can also be used explicitly

More on `public` & `private` ...

- The `public` & `private` keywords control the accessibility of members of a class, commonly known as **access modifiers**
- The `public` methods of a class are also known as the class's public services or public interface
- Clients of the class only need to concern the public interface of the class, and do not need to concern with how the class accomplishes its tasks
- `private` members, for internal use only!