CS1101 Programming and Problem Solving

Dr. Gina Bai

Spring 2023

Logistics

- **PA10 – A, B** on zyBook > Chap 11
  - Due: **Friday, April 7**, at 11:59pm

- **ZY-8B** on zyBook > Assignments
  - Due: **Wednesday, April 12**, at 11:59pm

- Midterm Exam 2 Regrade Request
  - Due: Tuesday, April 11

Array Sizes

zyBook Chap 8.7, 8.8

Recap – Motivating Example

Write a program that

- prompts user for number of students
- prompts user for Exam 1 grades for each student
- prints average grade for Exam 1
- prints number of students with Exam 1 grade higher than average

```
How many students? 5
Student 1’s Exam 1 Grade: 96
Student 2’s Exam 1 Grade: 92.5
Student 3’s Exam 1 Grade: 80.5
Student 4’s Exam 1 Grade: 99
Student 5’s Exam 1 Grade: 87
Average Exam 1 Grade: 91.0
3 students were above average.
```

Perfect Size Array Example

```java
import java.util.Scanner;

public class Gradebook {
    public static void main (String[] args) {
        Scanner input = new Scanner(System.in);
        System.out.print("How many students? ");

        while (!input.hasNextInt()){ // Verify user input
            input.next();  // Discard the invalid input
            System.out.print("How many students? (positive int) ");
        }
        int numStudents = input.nextInt();

        // Construct an array to store students' grades
        double[] exam1 = new double[numStudents];
        getGrades(exam1, input);

        double average = calcAvg(exam1);
        System.out.println("Average Exam 1 Grade = " + average);

        int numAbove = countAbove(exam1, average);
        System.out.println(numAbove + " students were above average.");
    }

    public static void getGrades (double[] exam1, Scanner input) {
        // Prompt user for Exam 1 grades for each student
        for (int i = 0; i < exam1.length; i++) {
            System.out.print("Student " + (i + 1) + "\'s Exam 1 Grade: ");
            while (!input.hasNextDouble()) { // Verify user input
                input.next();  // Discard the invalid input
                System.out.print("Student " + (i + 1) + "\'s Exam 1 Grade: ");
            }
            exam1[i] = input.nextDouble();
        }
    }
    // Other methods...
}
```

```
How many students? 5
Student 1’s Exam 1 Grade: 96
Student 2’s Exam 1 Grade: 92.5
Student 3’s Exam 1 Grade: 80.5
Student 4’s Exam 1 Grade: 99
Student 5’s Exam 1 Grade: 87
Average Exam 1 Grade: 91.0
3 students were above average.
```

Perfect Size Array Example

```java
import java.util.Scanner;

public class Gradebook {
    public static void main (String[] args) {
        Scanner input = new Scanner(System.in);
        System.out.print("How many students? ");

        while (!input.hasNextInt()){ // Verify user input
            input.next();  // Discard the invalid input
            System.out.print("How many students? (positive int) ");
        }
        int numStudents = input.nextInt();

        // Construct an array to store students' grades
        double[] exam1 = new double[numStudents];
        getGrades(exam1, input);

        double average = calcAvg(exam1);
        System.out.println("Average Exam 1 Grade = " + average);

        int numAbove = countAbove(exam1, average);
        System.out.println(numAbove + " students were above average.");
    }

    public static double calcAvg (double[] exam1) {
        double average = 0;
        for (int i = 0; i < exam1.length; i++) {
            average += exam1[i];
        }
        return average /= exam1.length;
    }

    // Other methods...
}
```

```
How many students? 5
Student 1’s Exam 1 Grade: 96
Student 2’s Exam 1 Grade: 92.5
Student 3’s Exam 1 Grade: 80.5
Student 4’s Exam 1 Grade: 99
Student 5’s Exam 1 Grade: 87
Average Exam 1 Grade: 91.0
3 students were above average.
```

Perfect Size Array Example

```java
import java.util.Scanner;

public class Gradebook {
    public static void main (String[] args) {
        Scanner input = new Scanner(System.in);
        System.out.print("How many students? ");

        while (!input.hasNextInt()){ // Verify user input
            input.next();  // Discard the invalid input
            System.out.print("How many students? (positive int) ");
        }
        int numStudents = input.nextInt();

        // Construct an array to store students' grades
        double[] exam1 = new double[numStudents];
        getGrades(exam1, input);

        double average = calcAvg(exam1);
        System.out.println("Average Exam 1 Grade = " + average);

        int numAbove = countAbove(exam1, average);
        System.out.println(numAbove + " students were above average.");
    }

    public static int countAbove (double[] exam1, double average) {
        int numAbove = 0;
        for (int i = 0; i < exam1.length; i++) {
            if (exam1[i] > average) {
                numAbove++;
            }
        }
        return numAbove;
    }
    // Other methods...
}
```

```
How many students? 5
Student 1’s Exam 1 Grade: 96
Student 2’s Exam 1 Grade: 92.5
Student 3’s Exam 1 Grade: 80.5
Student 4’s Exam 1 Grade: 99
Student 5’s Exam 1 Grade: 87
Average Exam 1 Grade: 91.0
3 students were above average.
```

Array Sizes

- Perfect size array
  - An array where the number of elements is **exactly equal to** the memory allocated.
- Oversize array
  - An array where the number of elements used is **less than or equal to** the memory allocated.
  - Since the number of elements used in an oversize array is usually less than the array's length, a **separate integer** variable is used to **keep track** of how many array elements are currently used.

Oversize Array Example

```
import java.util.Scanner;
import java.util.Arrays;

public class GradebookOversize {

    public static final int MAX = 20;

    public static void main (String[] args) {
        Scanner input = new Scanner(System.in);
        System.out.print("How many students? (less than " + MAX + ") ”);

        while (!input.hasNextInt()){ // Verify user input
            input.next();  // Discard the invalid input
            System.out.print("How many students? (positive int) ");
        }
        int numStudents = input.nextInt();

        // Construct an array to store students' grades
        double[] exam1 = new double[MAX];
        // LIVE CODING – ADD CODE HERE ...
    }

    // Other methods...
}
```

Oversize Array Example

```java
import java.util.Scanner;

public class GradebookOversize {
    public static final int MAX = 20;

    public static void main (String[] args) {
        Scanner input = new Scanner(System.in);
        System.out.print("How many students? (less than " + MAX + ") ");

        while (!input.hasNextInt()){ // Verify user input
            input.next();  // Discard the invalid input
            System.out.print("How many students? (positive int) ");
        }
        int numStudents = input.nextInt();

        // Construct an array to store students' grades
        double[] exam1 = new double[MAX];
        getGrades(exam1, input, numStudents);

        double average = calcAvg(exam1, numStudents);
        System.out.println("Average Exam 1 Grade = " + average);

        int numAbove = countAbove(exam1, average, numStudents);
        System.out.println(numAbove + " students were above average.");

    }

    public static void getGrades (double[] exam1, Scanner input, int numStudents) {
        // Prompt user for Exam 1 grades for each student
        for (int i = 0; i < numStudents; i++) {
            System.out.print("Student " + (i + 1) + "\'s Exam 1 Grade: ");
            while (!input.hasNextDouble()) { // Verify user input
                input.next();  // Discard the invalid input
                System.out.print("Student " + (i + 1) + "\'s Exam 1 Grade: ");
            }
            exam1[i] = input.nextDouble();
        }
    }
    // Other methods...
}
```

```
$ java GradebookOversize
How many students? (less than 20) 5
Student 1's Exam 1 Grade: 96
Student 2's Exam 1 Grade: 92.5
Student 3's Exam 1 Grade: 80.5
Student 4's Exam 1 Grade: 99
Student 5's Exam 1 Grade: 87
Average Exam 1 Grade = 91.0
3 students were above average.
```

Oversize Array Example

```java
import java.util.Scanner;

public class GradebookOversize {
    public static final int MAX = 20;

    public static void main (String[] args) {
        Scanner input = new Scanner(System.in);
        System.out.print("How many students? (less than " + MAX + ") ");

        while (!input.hasNextInt()){ // Verify user input
            input.next();  // Discard the invalid input
            System.out.print("How many students? (positive int) ");
        }
        int numStudents = input.nextInt();

        // Construct an array to store students' grades
        double[] exam1 = new double[MAX];
        getGrades(exam1, input, numStudents);

        double average = calcAvg(exam1, numStudents);
        System.out.println("Average Exam 1 Grade = " + average);

        int numAbove = countAbove(exam1, average, numStudents);
        System.out.println(numAbove + " students were above average.");

    }

    public static double calcAvg (double[] exam1, int numStudents) {
        double average = 0;
        for (int i = 0; i < numStudents; i++) {
            average += exam1[i];
        }
        return average /= numStudents;
    }
    // Other methods...
}
```

```
$ java GradebookOversize
How many students? (less than 20) 5
Student 1's Exam 1 Grade: 96
Student 2's Exam 1 Grade: 92.5
Student 3's Exam 1 Grade: 80.5
Student 4's Exam 1 Grade: 99
Student 5's Exam 1 Grade: 87
Average Exam 1 Grade = 91.0
3 students were above average.
```

Oversize Array Example

```java
import java.util.Scanner;

public class GradebookOversize {
    public static final int MAX = 20;
    public static void main (String[] args) {
        Scanner input = new Scanner(System.in);
        System.out.print("How many students? (less than " + MAX + ") ");

        while (!input.hasNextInt()){ // Verify user input
            input.next();  // Discard the invalid input
            System.out.print("How many students? (positive int) ");
        }
        int numStudents = input.nextInt();

        // Construct an array to store students' grades
        double[] exam1 = new double[MAX];
        getGrades(exam1, input, numStudents);

        double average = calcAvg(exam1, numStudents);
        System.out.println("Average Exam 1 Grade = " + average);

        int numAbove = countAbove(exam1, average, numStudents);
        System.out.println(numAbove + " students were above average.");

    }

    public static int countAbove (double[] exam1, double average, int numStudents) {
        int numAbove = 0;
        for (int i = 0; i < numStudents; i++) {
            if (exam1[i] > average) {
                numAbove++;
            }
        }
        return numAbove;
    }
    // Other methods...
}
```

```
$ java GradebookOversize
How many students? (less than 20) 5
Student 1's Exam 1 Grade: 96
Student 2's Exam 1 Grade: 92.5
Student 3's Exam 1 Grade: 80.5
Student 4's Exam 1 Grade: 99
Student 5's Exam 1 Grade: 87
Average Exam 1 Grade = 91.0
3 students were above average.
```

Object-Oriented Programming

Four Main Principles of OOP

1. **Abstraction**
2. **Encapsulation**
3. Inheritance
4. Polymorphism

Object Basics

zyBook Chap 9.1

Objects in the Real World

- Cars, books, computers, phones…
- Human beings
- Tickets, appointments, bank accounts…
- …

All of them have some sort of **data** and some **actions we can perform** with the data

Objects in the Program

Encapsulation of data and behavior

Object → A programming entity that contains…

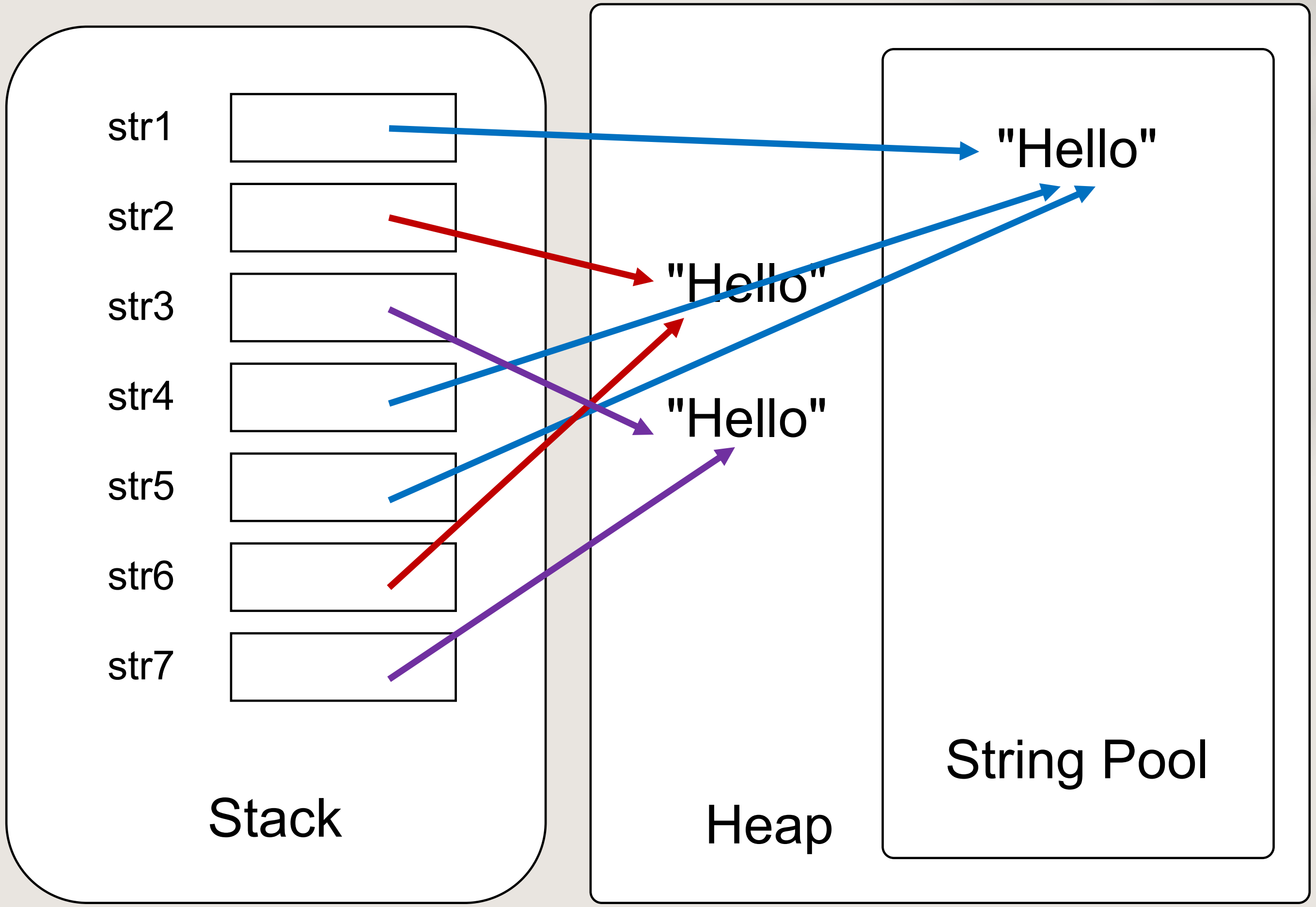
- **State (data)**
  - A set of **values (internal data)** stored in an object.
  - Represented by **fields/attributes/properties/instance variables** within the class.
- **Behavior (methods)**
  - A set of **actions** an object can **perform**, often reporting or modifying its internal state.
  - Represented by **instance methods** within the class.

Construct and Use an Object

- Construct an object:
  - `<objType> <objName> = new <objType>(parameters);`
    - `Scanner input = new Scanner(System.in);`
  - `int[] arr = new int[3];`
  - `String str = "Hello";`

(Optional)

```
String str1 = "Hello";
String str2 = new String("Hello");
String str3 = new String("Hello");
String str4 = "Hello";
String str5 = str1;
String str6 = str2;
String str7 = str3;
```



Construct and Use an Object

- Construct an object:
  - **`<objType> <objName> = new <objType>(parameters);`**
    - `Scanner input = new Scanner(System.in);`
  - `int[] arr = new int[3];`
  - `String str = "Hello";`
- Use an object’s field:
  - `arr.length;`
- Call an object's method:
  - `<objName>.<methodName>(parameters);`
    - `str.length();`

What defined an object? A Class!

Class → A program entity that represents…

- A program/module (a collection of procedures/actions), OR
- **A blueprint/template for a new type of object**

Blueprint Analogy

car1, car2, car3 are **instances** of the Car Class

Car Blueprint

State:
Make
Fuel Tank Capacity
Miles per Gallon
Gallons in Tank

Behavior:
Drive
Fill Gas Tank
Show Speed
Show Mileage
Show Fuel Level

car1

State:
Make: Acura
Fuel Tank Capacity: 21.0
Miles per Gallon: 18
Gallons in Tank: 18.4

Behavior:
Drive
Fill Gas Tank
Show Speed
Show Mileage
Show Fuel Level

car2

State:
Make: BMW
Fuel Tank Capacity: 21.9
Miles per Gallon: 21
Gallons in Tank: 9.8

Behavior:
Drive
Fill Gas Tank
Show Speed
Show Mileage
Show Fuel Level

car3

State:
Make: Cadilac
Fuel Tank Capacity: 19.0
Miles per Gallon: 24
Gallons in Tank: 12.1

Behavior:
Drive
Fill Gas Tank
Show Speed
Show Mileage
Show Fuel Level

How do we use objects in the real world?

- Such as computers, phones, cars, …
  - We need to learn ***how to use*** them given the instructions in the User Guide/Manual
  - We **DO NOT** need to know ***how they work***

How do we use objects in the program?

Client Program → A program that interacts with a class or objects of that class.

- Objects themselves are not complete programs
  - They are **components** that are given distinct roles and responsibilities
- Objects can be **used** and **reused** in many client programs to solve problems

Abstraction

Abstraction → Focus on **properties and external behaviors** rather than inner details.

- Objects from Java Class Library
    - We **understand the external behaviors** of these objects without knowing how they work
    - That’s why we need Javadoc ("User Manual/Guide") to introduce the object and explain how to use the methods

Creating Class – Define New Data/Object Type

- When creating our own new class, we are **abstracting the functionality** of the class for client programs

Object-Oriented Programming (OOP)

OOP is a modular approach where **data** and **functions** can be **combined** into a single unit known as an **object**

- It focuses on the objects that developers want to **manipulate** rather than the logic required to manipulate them
- It emphasizes **data** and **security** and provides the **reusability** of code

Fields and Instance Methods

zyBook Chap 9.2, 9.3, 9.4, 9.10, 9.11, 9.12, 9,13

Creating Class

Syntax for `<ClassName>.java`

```
public class <ClassName> {
    // ...
}
```

```java
// For example, the Book class in Book.java
public class Book {
    // ...
}
```

Access Modifiers/Specifiers

Modifier	Description
public	Accessible by self, derived classes, and everyone else.
private	Accessible by self.
protected	Accessible by self, derived classes, and other classes in the same package.
no specifier (default)	Accessible by self and other classes in the same package.

How to Achieve Abstraction?

Encapsulation → Hiding the implementation details of an object from the clients of the object, which leads to abstraction

- **Protect data from unwanted access**
- Clients cannot directly access or modify its internal workings – nor do they need to do so
- Allow us to change the internal workings of the class later without modifying client code

How to Encapsulate?

- Use **private** fields
  - **Visible inside the class** but are not visible outside the class

- Use **Accessor** and **Mutator** Methods
  - Write **accessor/getter methods** to **access** the private fields of a class
  - Write **mutator/setter methods** to **modify** the private fields of a class
  - Maintain encapsulation because class controls the access to internal data

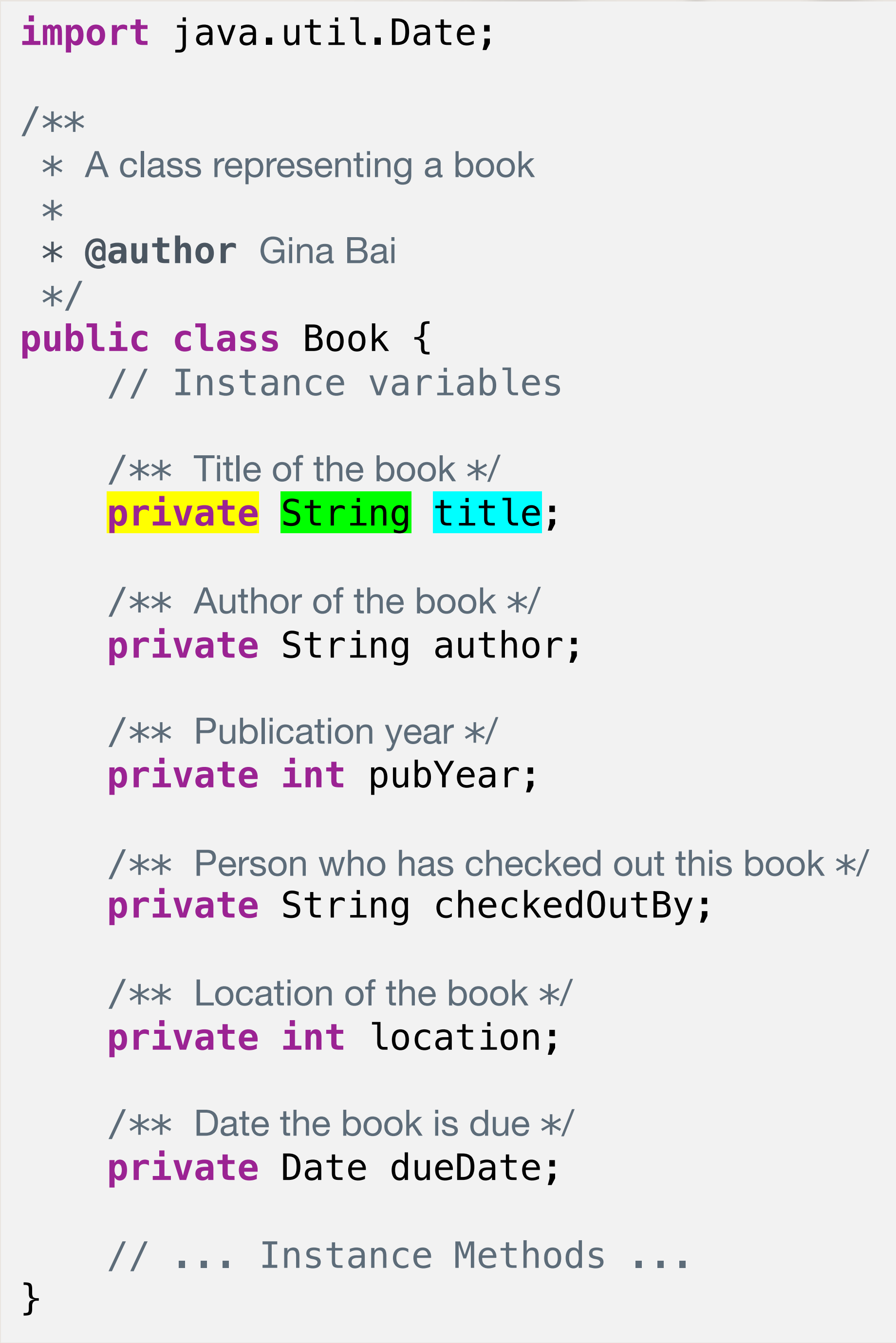
Fields/Instance Variables

Field → A variable **inside an object** that makes up part of its internal state

- Declaring a field **inside the class**, outside of all methods via:

  <accessModifier> <varType> <varName>

- Fields are given **default initial values** when an object is constructed
- Define the fields **at the top** of the class definition
  - Each object of this type will have these fields



```java
import java.util.Date;

/**
 * A class representing a book
 *
 * @author Gina Bai
 */
public class Book {
    // Instance variables

    /** Title of the book */
    private String title;

    /** Author of the book */
    private String author;

    /** Publication year */
    private int pubYear;

    /** Person who has checked out this book */
    private String checkedOutBy;

    /** Location of the book */
    private int location;

    /** Date the book is due */
    private Date dueDate;

    // ... Instance Methods ...
}
```

Instance Methods

Instance method → a method **inside an object** that operates on that object

- Instance methods are **called** on a specific object with the **dot notation**
- Instance methods can use the object's fields (object's fields have scope in the entire class)

```
public <returnType> <methodName>(<parameters>) {
    <statements>;
}
```

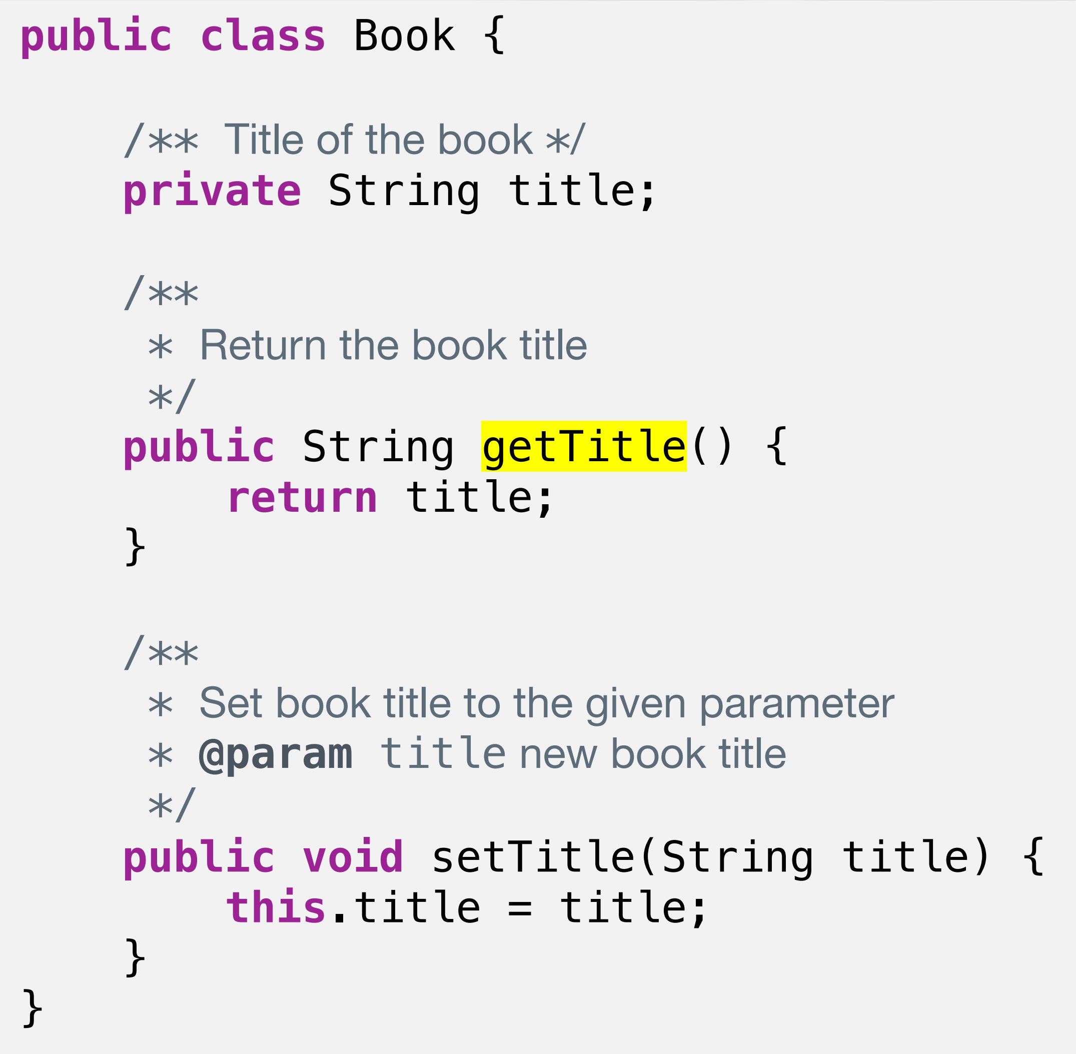
Recap – Static Methods vs. Instance Methods

- **Static Method**
  - A block of Java statements that is given a name.
  - Procedural decomposition

- **Instance Method**
  - A method inside an object that **operates on that object**.
  - Object-Oriented: the behavior is tied to the object

Method Types

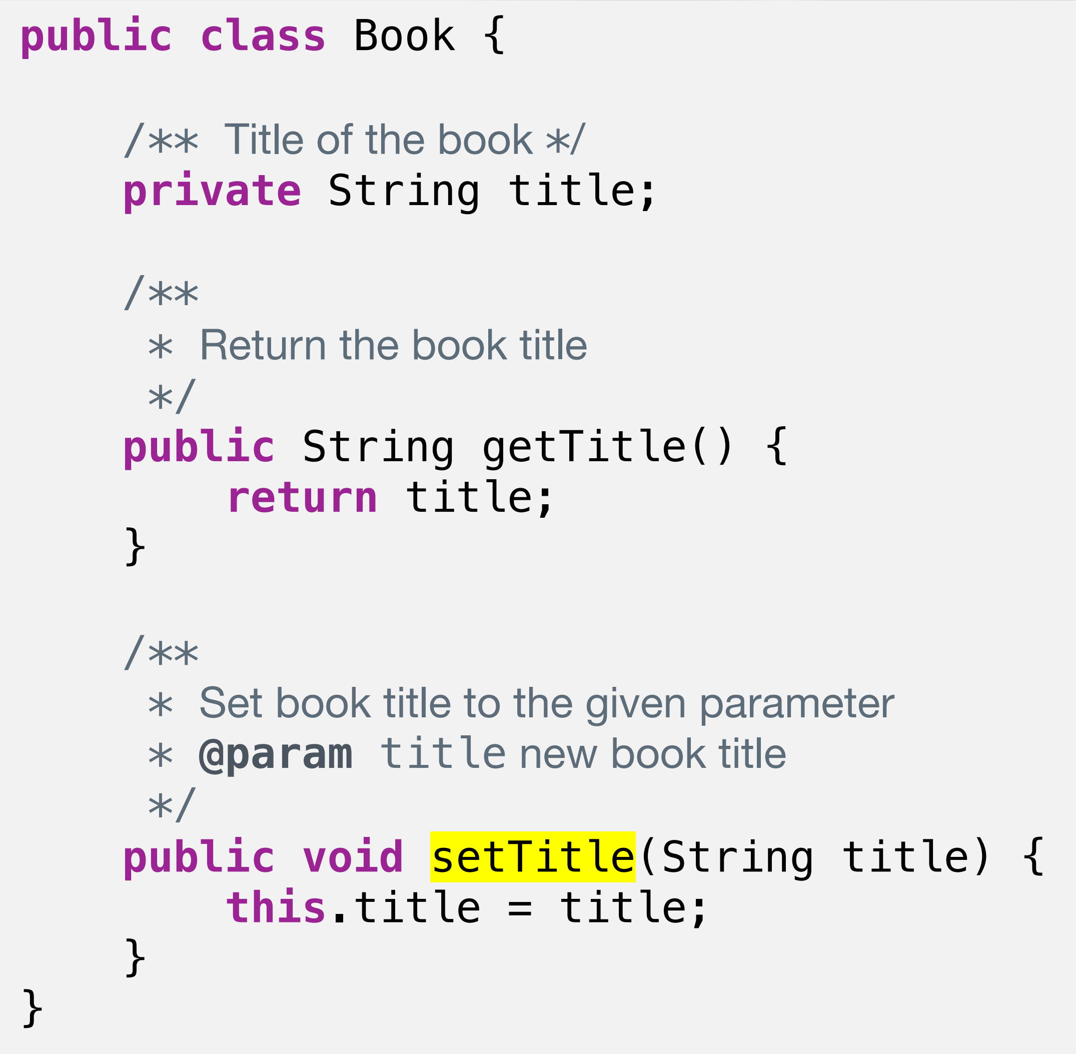
- **Accessor Method** → an instance method that provides information about the state of an object by returning the field or information about the field
  - **Getter method**
  - Commonly named as **get**FieldName()



```
public class Book {

    /** Title of the book */
    private String title;

    /**
     * Return the book title
     */
    public String getTitle() {
        return title;
    }

    /**
     * Set book title to the given parameter
     * @param title new book title
     */
    public void setTitle(String title) {
        this.title = title;
    }
}
```

Method Types

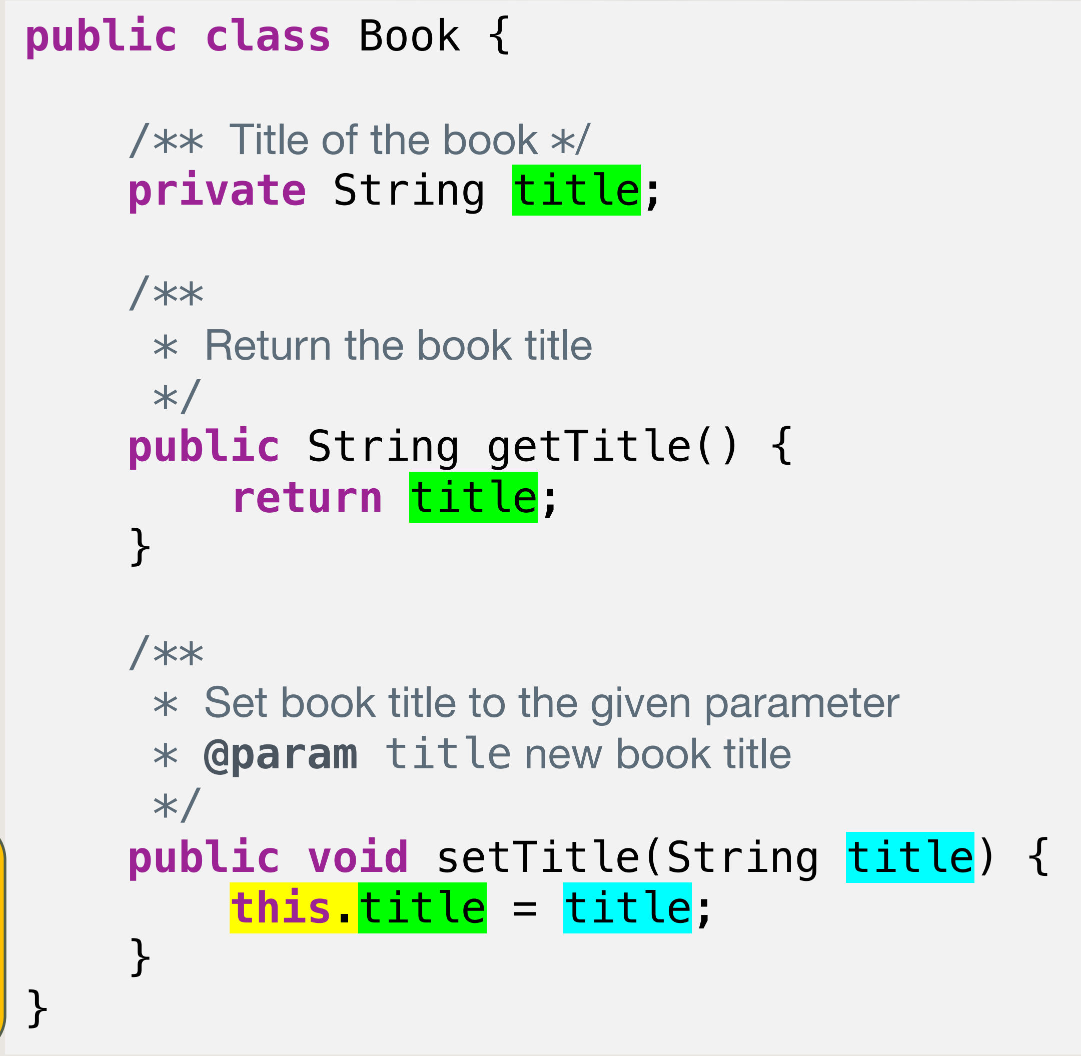
- **Mutator Method** → an instance method that modifies the object’s internal state
  - **Setter method**
  - Commonly named as **set**FieldName()



```java
public class Book {

    /** Title of the book */
    private String title;

    /**
     * Return the book title
     */
    public String getTitle() {
        return title;
    }

    /**
     * Set book title to the given parameter
     * @param title new book title
     */
    public void setTitle(String title) {
        this.title = title;
    }
}
```

Keyword – this

- Within an instance method or a constructor, the keyword this acts as a **special variable** that **holds a reference to the current object**, the object whose method or constructor is being called.



```java
public class Book {

    /** Title of the book */
    private String title;

    /**
     * Return the book title
     */
    public String getTitle() {
        return title;
    }

    /**
     * Set book title to the given parameter
     * @param title new book title
     */
    public void setTitle(String title) {
        this.title = title;
    }
}
```

this is **essential** if a **field** member and **parameter** have the **same identifier**

Implicit Parameter

- **Implicit Parameter** → The object that is referenced during an instance method call
  - For example, compiler views the object b in the following client code

```
Book b = new Book();
b.setTitle("Intro to Java Programming");
```

as the implicit parameter of the method call. That is:

```
setTitle(b, "Intro to Java Programming");
```

  - Within the instance method, we access the implicit parameter using the keyword **this**.

```
public class Book {

    /** Title of the book */
    private String title;

    /**
     * Return the book title
     */
    public String getTitle() {
        return title;
    }

    /**
     * Set book title to the given parameter
     * @param title new book title
     */
    public void setTitle(String title) {
        this.title = title;
    }
}
```