04_2 Constructors and Overloading

Object-Oriented Programming

Overloading

- Two or more methods (in the same class) have the same method name

```
public void setDate(int month, int day, int year)
public void setDate(String month, int day, int year)
public void setDate(int year)
```

- Any two definitions of the method name must have **different signatures**
  - Signature = (method name + parameter list)
  - Differing signatures: different numbers and/or types of parameters

```
setDate(int, int, int)
setDate(String, int, int)
setDate(int)
```

Example: Overloading (1/3)

```java
public class CarOverloading {
    public static void main(String[] args) {
        Car3 c1 = new Car3();
        Car3 c2 = new Car3();
        Car3 c3 = new Car3();
        c1.set("Hyundai Grandure", 2024);
        c2.set(2021);
        c3.set("Kia Niro");
        System.out.println("c1: " + c1);
        System.out.println("c2: " + c2);
        System.out.println("c3: " + c3);
    }
}
```

```
OUTPUT:
c1: Car3{model='Hyundai Grandure', year=2024}
c2: Car3{model='NO_MODEL', year=2021}
c3: Car3{model='Kia Niro', year=0}
```

Example: Overloading (2/3)

```
class Car3 {
    String model;
    int year;

    void set(String model, int year) {
        this.model = model;
        this.year = year;
    }
    void set(String model) {
        this.model = model;
        this.year = 0;
    }
```

```
    void set(int year) {
        this.model = "NO_MODEL";
        this.year = year;
    }
    void set() {
        this.model = "NO_MODEL";
        this.year = 0;
    }
```

Example: Overloading (3/3)

```java
    // equals method
    public boolean equals(Car3 other) {
        return model.equals(other.model) && (year == other.year);
    }


    // toString method
    public String toString() {
        return "Car3{" + "model='" + model + '\'' + ", year=" + year + '}';
    }
}
```

Cannot Overload Based on the Type Returned

- Return type is not the part of signature

```
public class SampleClass {
    public int computeSomething(int n) { ... }
    public double computeSomething(int n) { ... }  // Compile ERROR!!
}
```

Constructors

- Special kind of method
- Initializing the instance variables when object created
- Syntax: `public Classname(anyParameters) { code }`
  - A constructor must have the same name as the class
  - A constructor has no type returned, not even **`void`**
  - Constructors are typically overloaded

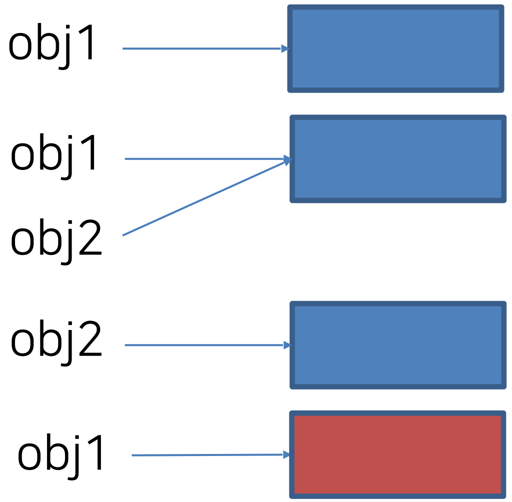
Calling Constructor

- Called when an object of the class is created using **new**

```
ClassName objectName = new ClassName(anyArgs);
```

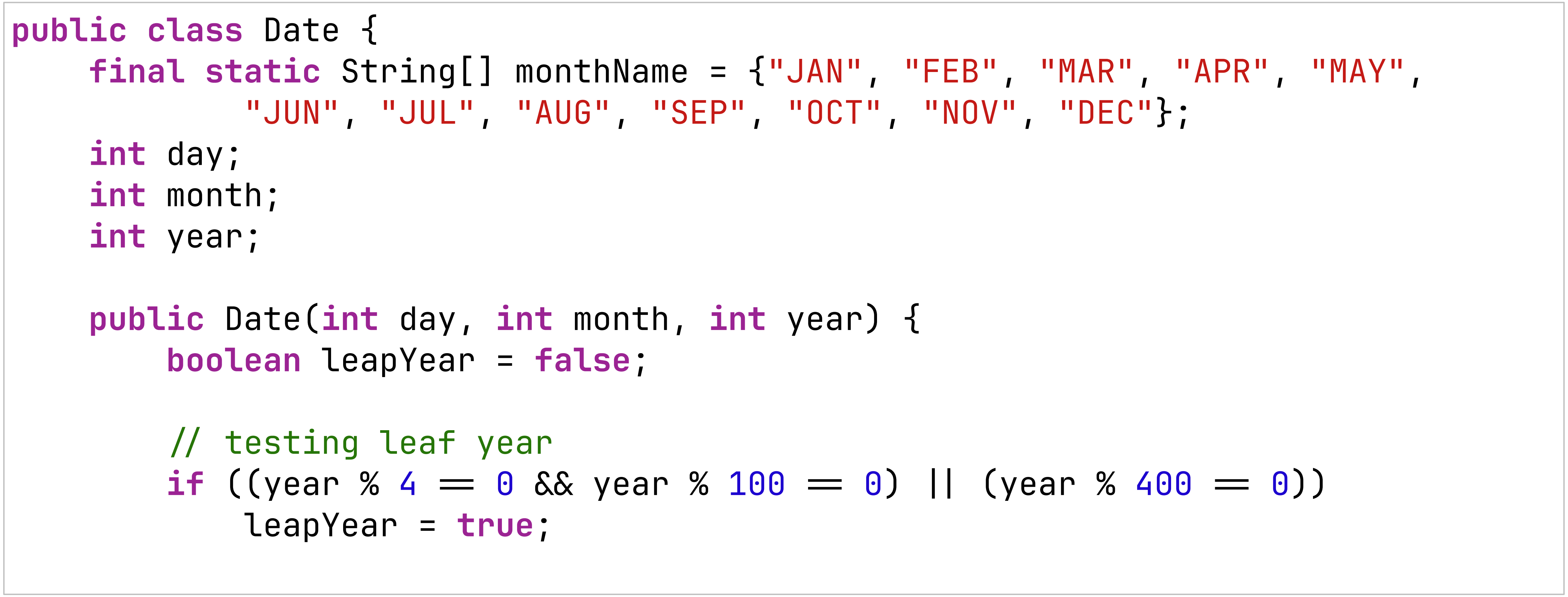
- If a constructor is invoked again (using **new**), the first object is discarded and an entirely new object is created

```
Class1 obj1 = new Class1(anyArgs);

Class1 obj2 = obj1;

obj1 = new Class1(anyArgs);
```



Example: Condition Test in Constructor (1/6)

- Test for the conditions that an instance variable should have (e.g., scope)



```java
public class Date {
    final static String[] monthName = {"JAN", "FEB", "MAR", "APR", "MAY",
            "JUN", "JUL", "AUG", "SEP", "OCT", "NOV", "DEC"};
    int day;
    int month;
    int year;

    public Date(int day, int month, int year) {
        boolean leapYear = false;

        // testing leaf year
        if ((year % 4 == 0 && year % 100 == 0) || (year % 400 == 0))
            leapYear = true;
```

Example: Condition Test in Constructor (2/6)

```
        // checking month
        if (month < 1 || month > 12) {
            if (month < 1) month = 1;
            else month = 12;
            System.out.print("Date Constructor: Wrong month < 1 or > 12");
            System.out.println(" month fixed to = " + month);
        }

        // checking day
        if (day < 1) {
            day = 1;
            System.out.print("Date Constructor: Wrong day < 1");
            System.out.println(" day fixed to = 1");
        }
```

Example: Condition Test in Constructor (3/6)

```java
        switch (month) {
            case 1:
            case 3:
            case 5:
            case 7:
            case 8:
            case 10:
            case 12:
                if (day > 31) {
                    System.out.print("Date Constructor: Wrong day > 31");
                    System.out.println(" day fixed to 31");
                    day = 31;
                }
                break;
```

Example: Condition Test in Constructor (4/6)

```
            case 2:
                if ((leapYear && day > 29)) {
                    System.out.print("Date Constructor:(Leap Year) Wrong day:" + day);
                    System.out.println(" day fixed to 29");
                    day = 29;
                }
                else if (!leapYear && day > 28) {
                    System.out.print("Date Constructor:(Normal Year) Wrong day:” +day);
                    System.out.println(" day fixed to 28");
                    day = 28;
                }
                break;
            default: // month = 4, 6, 9, 11
                if (day > 30) {
                    System.out.print("Date Constructor: Wrong day > 30");
                    System.out.println(" day fixed to 30");
                    day = 30;
                }
        }
```

Example: Condition Test in Constructor (5/6)

```
        this.year = year;
        this.month = month;
        this.day = day;
    }

    public String toString() {
        String acbc = "A.C.";
        if (year < 0) {
            year = -1 * year;
            acbc = "B.C.";
        }
        String answer = monthName[month - 1] + "." + day + ", " + year + " " + acbc;
        return answer;
    }
}
```

Example: Condition Test in Constructor (6/6)

```java
public class DateTest {
    public static void main(String[] args) {
        System.out.println("\n25, 4, -3295   ");
        Date date1 = new Date(25, 4, -3295);
        System.out.println(date1);


        System.out.println("\n29, 2, 1900   ");
        Date date2 = new Date(29, 2, 1900);
        System.out.println(date2);


        System.out.println("\n29, 2, 1898   ");
        Date date3 = new Date(29, 2, 1898);
        System.out.println(date3);


        System.out.println("\n5, -3, 1995   ");
        Date date4 = new Date(5, -3, 1995);
        System.out.println(date4);
    }
}
```

```
OUTPUT:
25, 4, -3295
APR.25, 3295 B.C.


29, 2, 1900
FEB.29, 1900 A.C.


29, 2, 1898
Date Constructor: (Normal Year) Wrong Feb. day: 29
day fixed to 28
FEB.28, 1898 A.C.

5, -3, 1995
Date Constructor: Wrong month < 1 or > 12 month
fixed to = 1
JAN.5, 1995 A.C.
```

A Constructor: `this` (1/2)

```java
public class ThisInConstructor {
    private String name;
    private int age;

    public ThisInConstructor() {    // Default constructor
        this("Unknown", 0); // Call another constructor
    }

    public ThisInConstructor(String name) {    // Constructor with name only
        this(name, 0); // Call another constructor
    }

    public ThisInConstructor(String name, int age) {// with name and age
        this.name = name;  // Here, 'this' is not for calling constructor
        this.age = age;
    }
```

A Constructor: `this` (2/2)

```
    public String toString() {
        return "Name: " + name + ", Age: " + age;
    }

    public static void main(String[] args) {
        ThisInConstructor person1 = new ThisInConstructor();
        System.out.println("Person1: " + person1);

        ThisInConstructor person2 = new ThisInConstructor("Alice");
        System.out.println("Person2: " + person2);

        ThisInConstructor person3 = new ThisInConstructor("Bob", 30);
        System.out.println("Person3: " + person3);
    }
}
```

```
OUTPUT:
Person1: Name: Unknown, Age: 0
Person2: Name: Alice, Age: 0
Person3: Name: Bob, Age: 30
```

Provide Your Default Constructor! (1/2)

- Default Constructor
  - A constructor having no argument, ex) Date x = new Date();

- If no constructor in the class, Java will **automatically create a default constructor** (but doesn't do anything).

- If one or more constructor exist in the class, Java will **not provide default constructor**.

Provide Your Default Constructor! (2/2)

```
public class Date2Test {
    public static void main(String[] args) {
        Date2 d = new Date2();  // ERROR! no default constructor provided
    }
}

class Date2 {
    private int month;
    private int day;
    private int year;

    public Date2(int month, int day, int year) { // user-written constructor
        this.month = month;
        this.day = day;
        this.year = year;
    }
}
```

Default Variable Initializations (1/2)

- Instance variables are automatically initialized
  - `boolean` types are initialized to false
  - Other primitives are initialized to the zero of their type
  - Class types are initialized to `null`
- However, it is a better practice to explicitly initialize instance variables in a constructor
- Note: Local variables are not automatically initialized

Default Variable Initializations (2/2)

```
public class ATest4 {
    public static void main(String[] args) {
        int x, y;
        FooClass f = new FooClass();
        System.out.println(f.a + " " + f.b + " " + f.c);
        System.out.println(x + " " + y); // ERROR!! x, y not initialized
    }
}

class FooClass {
    int a;
    boolean b;
    double c;
}

OUTPUT:
0 false 0.0
```

Example: StringTokenizer Class

```
import java.util.StringTokenizer;

public class StringTokenizerDemo {
    public static void main(String[] args) {
        // Example input string
        String input = "Java is a high-level, class-based; object-oriented programming
language.";

        // Creating a StringTokenizer with space, comma, and semicolon as delimiters
        StringTokenizer tokenizer = new StringTokenizer(input, " ,;");

        // Counting tokens
        int tokenCount = tokenizer.countTokens();
        System.out.println("Total number of tokens: " + tokenCount);

        // Iterating through tokens and printing each token
        while (tokenizer.hasMoreTokens()) {
            String token = tokenizer.nextToken();
            System.out.println(token);
        }
    }
}
```

Example: OUTPUT

```
"Java is a high-level, class-based; object-oriented programming language.";
```

```
Java
is
a
high-level
class-based
object-oriented
programming
language.
```