Java Objects and Classes

A BPI Training Program

About Me

James Paolo W. Menguito

Software Engineer - 7YRS

Exist Software Labs, Inc.

Caishen - 3YRS

What You'll Learn

1. Objects and Classes
2. Object-Oriented Programming
3. Advanced Java

What we'll cover today

1. Objects
2. Classes
3. Variables
4. Constructors
5. Methods

Review

OBJECT

- represents real-world entities, data, or concepts in your program
- an instance of a class and encapsulates both data and behaviors
- core of the OOP paradigm
- Example: a class Person and an instance of a Person is me (or you)
- *** Explain “instance”

OBJECT - Key Details

- objects are created based on class definition
- each object has the same structure of a class but can have different data
- the data pertains to the attributes/fields defined by the class, these represent the state of the object.
- methods are functions or behaviors the object can take.

Creating an Object

```
public static void main(String[] args) {
    // create an object james of the class "Person"
    Person james = new Person();
}
```

CLASSES

- serves as a blueprint or a template for creating objects
- defines structure and behavior
- class name is written using PascalCase
- Example: define what a class Person has (name, age, weight, height, move(), speak())

Defining a Class

```
public class Person {
    String name;     // attribute
    int age;         // attribute

    void speak(){    // method
        System.out.print("My name is " + name);
    }

    void walk() {
        System.out.print("I am now walking.");
    }
}
```

Questions?

- //

Seatwork 1 - Individual

- identify at least one attribute and at least one method for the ff classes

```
public class Book {

}
```

```
public class House {

}
```

```
public class Tree {

}
```

Instance Variables

- Also known as fields / attributes
- Can have a default value based on class definition
- Ideally declared as “private” (will dive more into this later)
- “What the object knows”

Instance Variables - Example 1

```
public class Person {
    String name;     // "James"
    int age;         // 25
    float weight;    // example value 91.2

    void speak(){    // method
```

Instance Variables - Example 2

```
String name;    // "James"

// method
void speak() {
    // "this" keyword is a "reference variable"
    // referring to the current object
    System.out.println("My name is " + this.name);
}

void setName(String name) {
    this.name = name; // "this" keyword eliminates ambiguity
}
```

Constructor

- A special method that is called when creating an object (instantiation)
- Called in combination with the “new” keyword
- Must have the same name as the class
- Parameters are optional
- No return type
- Can be used to initialize instance variables upon object creation

Constructor - Example 1

- Real world
- When you are born, you are given a name are your weight is determined

Constructor - Example 2

```
public static void main(String[] args) {
    // create an object james of the class "Person"
    Person james = new Person();

}
```

3 Types of Constructors

1. Default Constructor
2. No-Argument Constructor
3. Parameterized Constructor

Default Constructor

- Created implicitly by Java when no constructor is defined in the class.
- Example: Person {} and call new Person()

Default Constructor

```
public class Person {
    String name;    // "James"
    int age;        // 25
    float weight;   // example value 91.2

    // no constructor declared here!

    void walk() {
        System.out.println("I am now walking.");
    }
```

```
public class MyApplication {

    public static void main(String[] args) {
        // create james
        Person james = new Person();// default constructor
        james.name = "James";       // set name to James
        james.speak();              // call speak to output name

    }
```

```
<terminated> MyApplication
My name is James
```

No-Argument Constructor

- A constructor with no parameters
- Used when values are initialized within the class
- Example: define Person() {} inside Person class but provide default values for the fields. It doesn't have to make sense since this is a program.

No-Argument Constructor

```
public class Person {

    String name;    // "James"
    int age;        // 25
    float weight;   // example value 91.2

    // no-args constructor
    public Person() {

    }
public class MyApplication {

    public static void main(String[] args) {
        // create james
        Person james = new Person();// default constructor
        james.name = "James";       // set name to James
        james.speak();              // call speak to output name

    }
```

```
<terminated> MyApplication
My name is James
```

Parameterized Constructor

- Used to initialize values
- Dynamic Values are initialized upon creation.
- Example: define Person(String name, float age)

Parameterized Constructor

```java
public class Person {

    String name;    // "James"
    int age;        // 25
    float weight;   // example value 91.2

    // parameterized constructor
    public Person(String name, int age, float weight) {
        this.name = name;
        this.age = age;
        this.weight = weight;
    }
```

```java
public class MyApplication {

    public static void main(String[] args) {
        // create james
        Person james = new Person("Jameszu", 300, 150.0f);

        james.setName("James");
        System.out.print("This person's name is " + james.getName() );

    }

}
```

METHODS

- defines actions or behaviors
- encapsulates functionality
- a set of statements
- can be used to set attributes or get attributes
- “things the object does”

Declaring Methods / Method Declaration

- specify the method's name, the return type, and a list of parameters (optional)
- the return type indicates the data type of the value the method can return

Method Signature

- Refers to the combination of the method name and the parameter list

Method Body

- the actual logic of the method that is executed when a method is called
- contains the block of code enclosed with curly braces

Local Variables

- A variable declared inside the method body
- Only exists within the method and does not belong to the class itself
- May serve different purposes such as a temporary holder for a value

Method Invocation

- use a method by calling it by its name and providing any parameters (if any)
- typically done on an object using the “dot” operator (.)

Method - Example

- Method Example: given a class Person, create a new method for the class Person and provide parameters like String sleep(String placeToSleep, int hours) and return result “Well Rested” or “Still sleepy” based on the hours slept. Demonstrate use of local variables.

Method - Example

```
String sleep(String placeToSleep, int hours) {
    int wellRestedSleepHours = 8;
    String result = null;

    if(hours >= wellRestedSleepHours) {
        result = "I am well rested, I slept at " + placeToSleep;
    } else if (hours < wellRestedSleepHours) {
        result = "I did not sleep well, I slept at " + placeToSleep;
    }
    return result;
}
```

Seatwork 2 - Individual

- Complete the code snippet below to achieve the output

```
public class WhiteBoard {


    int addNumbers(int addend1, int addend2){
        <fill> = addend1 + addend2;
        <fill>;
    }
}

public class SeatWork2 {
    public static void main(String[] args) {

        WhiteBoard wb = new WhiteBoard();
        <fill> = wb.addNumbers(5,3);
        System.out.println("The total is: " + String.valueOf(<fill>) );
    }
}
```

```
<terminated> SeatWork2
The total is: 8
```

VARIABLES

- used to store and manage data
- data manipulated by the application

Declaring Variables / Variable Declaration

- Indicate the data type and declare the name of the variable
- Variable names should be typically written in camelCase

Initializing Variables

- Assign a value to the variable upon declaration
- Primitive data types typically have a default value of 0 while objects have a default value of null
- Uses the “equal” sign to assign values
- Example: int age = 10;

Variable Scope

- where a variable can be accessed

Variable Lifetime

- Until when does a variable exist in memory

Types of Variables

1. Local variables
2. Instance variables
3. Static Variables (Class variables)

Local Variables

- Scope: inside a code block such as a constructor or method or other control structures like loops and can only be accessed in that specific code block
- Lifetime: exists until the end of the code block or the method execution
- Example: mention example above.

Instance Variables

- Also known as fields / attributes
- Scope: declared in a class but outside of methods and can be accessed anywhere within the class
- Lifetime: exists as long as the object it belongs to exists in memory
- Example: Person's name, and age

Static Variables (Class variables)

- Declared using the keyword “static” inside the class but outside of methods
- Scope: Belongs to the class itself and only one value is accessible by class
- Lifetime: exists as long as the class is loaded in memory
- Example: Person class and static field race = HUMAN

Static Variables (Class variables)

Establishing the use of Getters and Setters

- Using the access modifier “private” is best practice for attributes
- Getters and Setters are public methods that are used to prevent direct access of the private attributes
- Getters and Setters and the use of “private” promotes Object Oriented Programming concepts by enforcing data hiding and controlling data access
- This will be discussed further in OOP.

Object Identity and References

- All objects have a unique identity even if they have the same data
- Objects are referred to using References
- Multiple references can point to a single object
- When an object is created using “new”, memory is allocated on the heap for the object's instance variables

Exercise 1 - Individual

1. Students must create a class Car
2. Students must add attributes
3. Students must create at least one method
4. Students must provide create a Parameterized Constructor
5. Students must be able to instantiate (create) two cars. One with No Args and One using Parameterized Constructor
6. Students must be able to assign values to the attributes.
7. Students must be able to call the method

Questions?

- //