Classes and Objects

Table of Contents

1. Describe the difference between the terms *class* and *objects*.

    - **class**: this is a abstract data type with attributes and methods
    - **object**: this is an instance of a class

2. Label the different parts of the following class:

```java
public class Book {                                 // <- class definition: singular, capitalised
    private String title;                           // <- class attributes
    private String author;                          // <- default value is null if not specified
    private String borrowedBy = null;               // <- default values for attributes
    private boolean borrowed = false;
    private int borrowDuration;                     // <- default value is 0

    public Book(String author, String title) {  // <- constructor
        this.author= author;
        this.title = title;
    }

    public void borrow(String owner, int duration) { // <- class method
        borrowed = true;
        borrowedBy = owner;
        borrowDuration = duration;
    }
}
```

3. What is the purpose of a constructor, and how do we use them?

    - A constructor is used to create and initialise an object
    - e.g. to initialise a new book: `Book book = new Book("James", 14);`

4. What does the keyword `this` mean? Why do we use it?

    - `this` refers to the calling object
    - used to refer to attributes/methods of the calling object, e.g. in constructors so that you can use the same name for the constructor argument and the attribute

- sometimes people use `_` as a prefix to the argument name so that you don't need to use this

5. What does **`null`** mean in Java?

   - it's a constant that can be assigned to any data type in Java, indicating the variable has no real value
   - can be used to initialise variables where there is no obvious/useful choice
   - **`null`** is not an object: for comparison you use normal operators `==` `!=`, not `equals` method
   - attempting to invoke a method on a **`null`** object will throw a *Null Pointer Exception*

6. For the following questions, the class definition for `IntegerHolder` is:

```java
class IntegerHolder {
    int value;
    public IntegerHolder(int value) {
        this.value = value;
    }
}
```

Determine the output for each code snippet. a.

```java
public static void increment(int input) {
    input = input + 1;
}
public static void main(String[] args) {
    int a = 0;
    increment(a);
    System.out.println(a);  // prints "0" as no value is returned, and
       no reference to a is passed, int is passed by value
}
```

b.

```java
public static void triple(IntegerHolder integerHolder) {
    integerHolder.value = integerHolder.value * 3;
}
public static void main(String[] args) {
    int a = 25;
    IntegerHolder myHolder = new IntegerHolder(a);
    triple(myHolder);
    System.out.println(myHolder.value); // prints "75"
    System.out.println(a);  // prints "25"
}
```

7. What are getters and setters in Java? Why are they needed?

- getters/setters are used to mutate state of an object
- access control: ensures you are modifying object per prescribed behaviour: produces a more secure/predictable result
- you define a clean interface with which to interact/act upon an object
- hides implementation details

8. What are two special methods that every class in Java has? What do they do? (Hint: not getters/setters)

   - `equals()`: allows you to make equality comparison between two objects
   - `toString()`: allows you to print a string representation of an object
   - `clone()`: produce a copy of an object

9. Static attributes and methods

   - shared between all instances of a class
   - c.f. global variables in C
   - easy to write confusing/difficult to maintain code
   - occassionally they are the write thing to do
   - for variables in a method (not attributes!) you do not use **`private`** keyword
   - non-static attributes/methods end up on heap (dynamic memory)
   - static attributes/methods end up in static memory (similar to stack)
   - useful for e.g. counting number of instances of a given class
   - `System.out.println("Hello");` // out is a static attribute of System
   - `Math.sqrt(2.0);` // sqrt() is a static method of Math
   - be aware compiler will say “Did you want this to be a static attribute?” when you try to reference a non-static attribute without an instance reference

Design a chair class

- attributes
  - number of legs
  - material
  - height
  - price
  - manufacturer
  - owner
  - chair is occupied
- methods

  - get/set attribute

Complex number

- attribute
  - real
  - imaginary
- methods
  - set real
  - set imaginary
  - get real
  - get imaginary
  - equals
  - toString
  - modulus
  - angle

```java
public class ComplexNumber {
    private double real;
    private double imaginary;

    public ComplexNumber(double real, double imaginary) {
        this.real = real;
        this.imaginary = imaginary;
    }

    public double getReal() {
        return real;
    }

    public double getImaginary() {
        return imaginary;
    }

    public void setReal(double real) {
        this.real = real;
    }

    public void setImaginary(double imaginary) {
        this.imaginary = imaginary;
    }

    public double getModulus() {
        return Math.sqrt(Math.pow(real, 2) + Math.pow(imaginary, 2));
```

```
28      }
29
30      public boolean equals(ComplexNumber c) {
31          return Double.compare(this.real, c.real) == 0 && Double.compare
                (this.imaginary, c.imaginary) == 0;
32      }
33  }
```

Can a class have multiple parent classes? - Java says no, diamond problem (see wiki