CODING BLACK FEMALES

Introduction to Java

Agenda:

Topics:

- What is Java?
- How is it used in industry
- Programming Concepts in Java
  - Loops
  - Methods
  - Conditional Statements
  - Variables
  - Arrays
- Introduction to Object Oriented Programming
- Object Oriented Programming in Java:
  - Classes
  - Inheritance
  - Abstraction
  - Encapsulation

What is Java?

What is Java?

Java is a programming language:

- Java is a programming language and computing platform first released by Sun Microsystems in 1995.
- Java is a general-purpose programming language, it can be used to create all kinds of programs.
- From laptops to datacenters, game consoles to scientific supercomputers, cell phones to the Internet, Java is everywhere!

Uses of Java in Industry

Uses of Java in Industry

Java can be used in a variety of contexts:

- Mobile App Development.
- Desktop GUI Applications.
- Web-based Applications.
- Gaming Applications.
- Big Data Technologies.
- Distributed Applications.
- Cloud-based Applications.
- IoT Applications.

Uses of Java in Industry

Fun facts about Java:

- 97% of Enterprise Desktops Run Java
- 89% of Desktops (or Computers) in the U.S. Run Java
- 9 Million Java Developers Worldwide
- #1 Choice for Developers
- #1 Development Platform
- 3 Billion Mobile Phones Run Java
- 100% of Blu-ray Disc Players Ship with Java
- 125 million TV devices run Java

Uses of Java in Industry

Java can be used in a variety of industries:

- Over 90% of the Fortune 500 companies still use Java for their development projects.

- Globally, there are over 8 million Java developers

Variables

Variables

A variable can be seen as a box:

- A variable in Java and many other programming languages is like your everyday box.
- The box is labelled so you don't get mixed up and you can always refer to the contents of the box using its label.
- You can also remove, change or add elements to the box.

Variables

- In this analogy the box is a memory location in your computer, and the label is the name you give the memory location when you program.

- In this workshop we will go over how you can create and use variables, and all the different types of variables you can use.

Variables

Limitations to Variable names:

When creating a variable, you have to specify its name. There are some rules to the name you can give a variable:

1. Must start with letter or _
2. Must only contain alphanumeric and _
3. There are also a bunch of words that are reserved in Java, this means they are used by java itself! so you cant use them. For example ‘if’, ‘true’ or ‘false’.

Variables

In Java, every variable must be given a type
There are 8 common basic types in Java:

1. byte: A much shorter number (-128 to 127)
2. short: A short number (-32768 to 32767)
3. int: A number (-2,147,483,648 to 2,147,483,647)
4. long: A large number (-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807)

Variables

5. float: A small decimal
6. double: A larger decimal
7. char: A letter/symbol
8. boolean: true or false

Strings are also a common type but it is not a basic type. A string is just text written directly using " ". This is essentially just a sequence of letters (char).

Example Code:

Variables

```
int dogAge = 12
String name = “Chib”
float pi = 3.14
```

Examples: defining Variables.

How are you feeling?

RED
I have no idea what you're talking about

YELLOW
I have some questions but feel like I understand some things

GREEN
I feel comfortable with everything you've said

Conditional Statements

Conditional Statements

What are Conditional Statements?

- There are some cases in real life where the action we want to take is directly dependent on some situation or condition.
- An example is decision situation of whether you should take an umbrella or not. The answer (‘yes’ or ‘no’) depends on the condition of there being rain.
- *If it is raining then take an umbrella else, don’t take one.*

Conditional Statement:

The simplest conditional statement in java is the basic 'if' statement.

If Statement

```
if (condition) {
  // body - the action to perform when
the condition is met
  // e.g. printing "take an umbrella"
 }
```

Conditional Statement:

On the right we see:

If [it is raining - condition] then [take an umbrella - body/action].

If it is not raining, nothing happens.

If Statement

```
boolean raining = true;

if (raining) {
  // Since raining is set to true, the body is run
   System.out.println("Take an umbrella");
}
```

Conditional Statement:

On the right we see an If-Else statement:

If [it is raining - condition] then [take an umbrella] else [do not take an umbrella]

If-else Statement

```
boolean raining = true;

if (raining) {
  // If raining variable was set to false, this would not run
  System.out.println("Take an umbrella");
} else {
   System.out.println("Do not take an umbrella");
   // The line above is run if raining is set to false
}
```

How are you feeling?

RED
I have no idea what you're talking about

YELLOW
I have some questions but feel like I understand some things

GREEN
I feel comfortable with everything you've said

Loops

For Loops

We will look at how to repeat some actions:

- A ‘for loop’ is also known as a definite loop.
- It is a looping statement used when we know the range of values we wanted to loop over e.g. from 1 to 1000
- A certain action is repeated for a specified number of times

For Loops

For loop: the variable ‘i’ is a counter that is used to keep track of the number loops that have been done.

It does not need to be ‘i’, the variable can be called anything.

Loops

```
for (i = 1; i <= 1000; i++){ // Go over all the
numbers from 1 to 1000
  if (i % 2 == 0){ // if the number is divisible by 2
    System.out.println(i); // print the number
  }
}
```

While Loops

We will look at how to repeat some actions:

- A ‘while loop’ is known as an indefinite loop.
- It is used when we do not know how long we should loop for but we intend to keep going until some loop condition is broken.
- In the case of a while loop, the loop condition does not have to use a counter
- e.g. *while it is raining, keep the umbrella open.*
- The condition making us do the action is the fact that it is raining. Once this is broken, the loop stops.

While Loops

While Loop: we create a counter (called ‘count’, in this example). While the count variable is less than 1000, the if-statement is performed. The count variable then increases by 1, and the cycle continues.

The While loop stops when the count variable is bigger than 1000.

While Loop

```
int count = 1; // Start counting from 1

while (count <= 1000){ // while the current number is less
or equal to 1000
  if (i % 2 == 0){ // if the number is divisible by 2
    System.out.println(i); // print the number
  }
  count++; // Move on to the next number
}
```

While vs For Loop

How to choose between ‘While’ and ‘For’

- Choose a for loop when you know the exact number of times that you want to loop for
- Use a while loop if you have a condition that must be met (or broken) to stop the loop

Example:
For loop: If you want to display a weather report for each day of the week -> loop a function exactly 7 times.

While loop: If you want to keep asking for the user’s password until they enter it correctly -> while their password is wrong, keep asking for the correct one.

Arrays

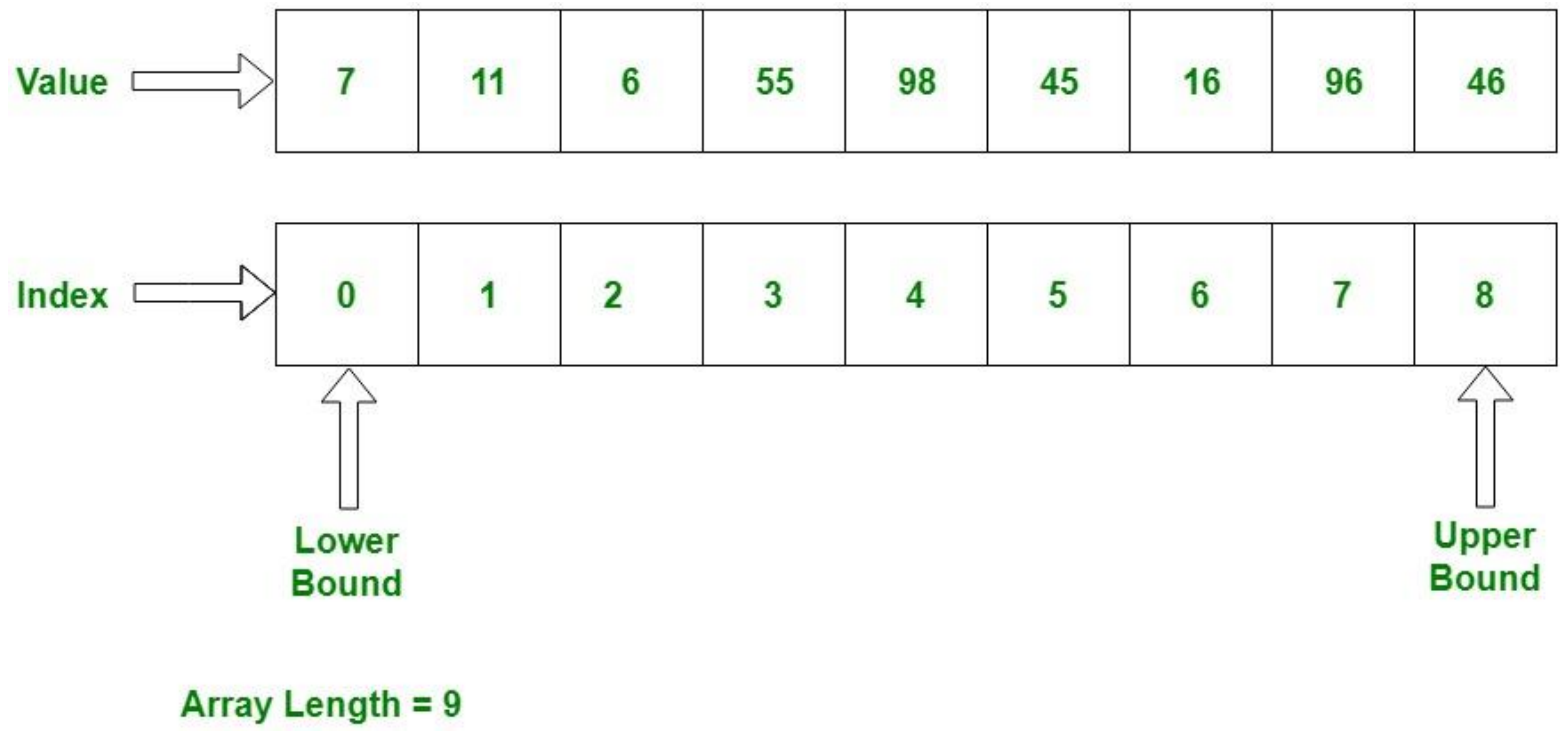
Arrays

What they are for

- Arrays can be used to represent a collection of variables or values of the same type

How they work

- Arrays are indexed from 0, have a length.
- Each entry can be read or written



Arrays

Define an array

The size of the array must be known!

Array Initialization

```
// array of integers
int[] myIntArray = {1,2,3}
int[] emptyIntArray = new int[4];
int[] myIntArray = new int[]{1,3,3}
double[] myDoubleArray = {1.0, 2.0, 3.0}
int [] myIntArray = {2,3}
int []myIntArray = {5,6}
int myIntArray[] = {0,2}

machineSlot[] machineSlots = new
machineSlot[NUM_OBJECTS];
```

Arrays

Read and Write array items

Array index starts at zero!

Array Operations

```
// array of integers
int[] myIntArray = {1,2,3}
// prints 3
 System.out.println(myIntArray[2]);
// sets first element of array to 4

myIntArray[0] = 4;
```

Arrays

Loop over array items

Make sure to stop reading after the end of the array!

Array Operations

```
// array of integers
int[] myIntArray = {1,2,3};
// prints 1 2 3
for (int i = 0; i < myIntArray.length; i++) {
  System.out.print(myIntArray[i]);
}
// also prints 1 2 3, For-Each

for (int i : myIntArray) {

  System.out.print(i + " ");

}
```

Arrays

Multi-dimensional Arrays

Arrays can be 2D, 3D, 4D, …

Array Operations

```
int[][] myIntArray = {{1,2},{3,4}};

int[][][] myIntArray =

   {{{1,2},{3,4}},{{1,2},{3,4}}};
```

How are you feeling?

RED
I have no idea what you're talking about

YELLOW
I have some questions but feel like I understand some things

GREEN
I feel comfortable with everything you've said

Exercise 1: Programming in Java

Exercise 1: Create a Vending Machine

The program should :

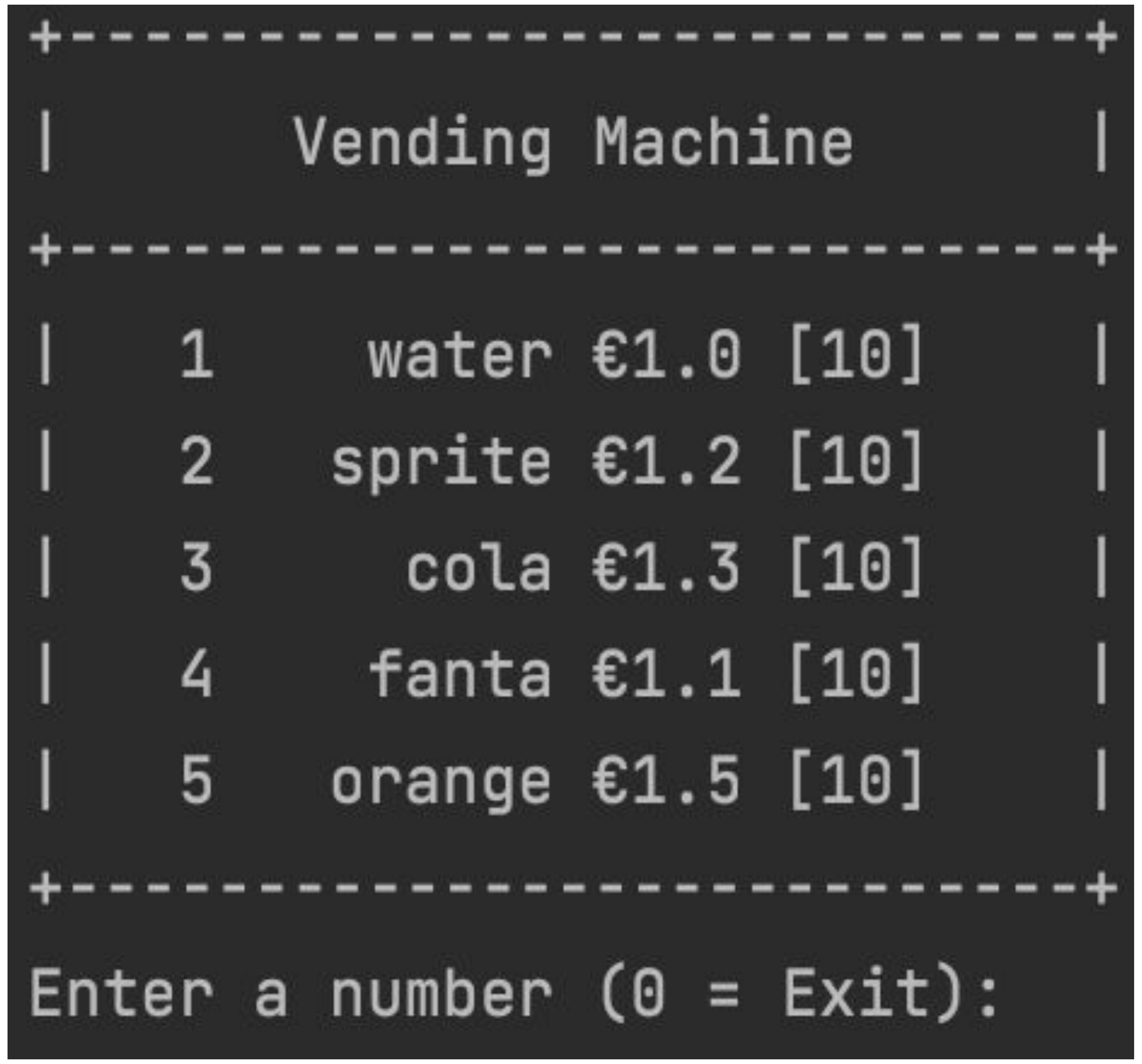
- show the contents of the vending machine
- prompt the user to ask which item to purchase
- update stock to show user if the machine is out of an item
- run in a loop so that you may purchase multiple items
- allow the user to exit the program and show the bill

Exercise 1: Create a Vending Machine

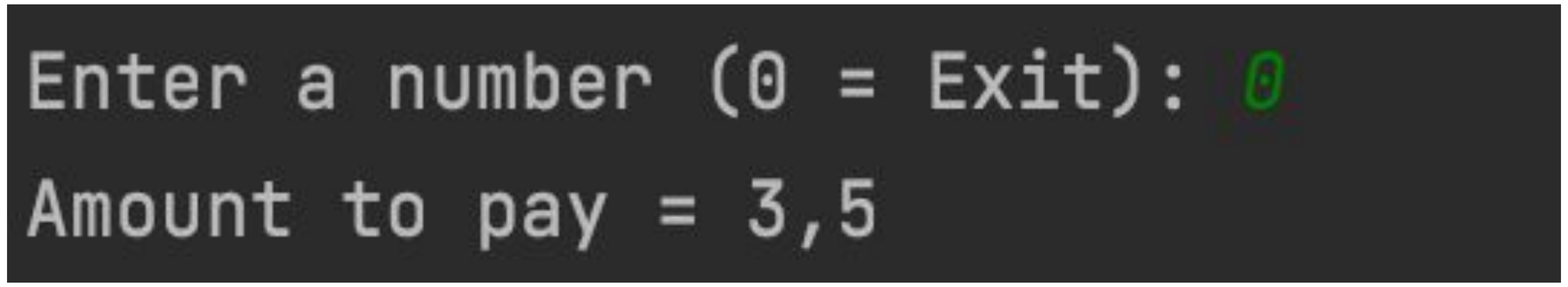
- How we represent the data : Choosing data types
- How we interact with the user : Use standard functions
- How we group instructions : Building functions
- How we create logic in functions : Using control flow
- How we make it work : Putting it all together

Exercise 1: Create a Vending Machine

An example:



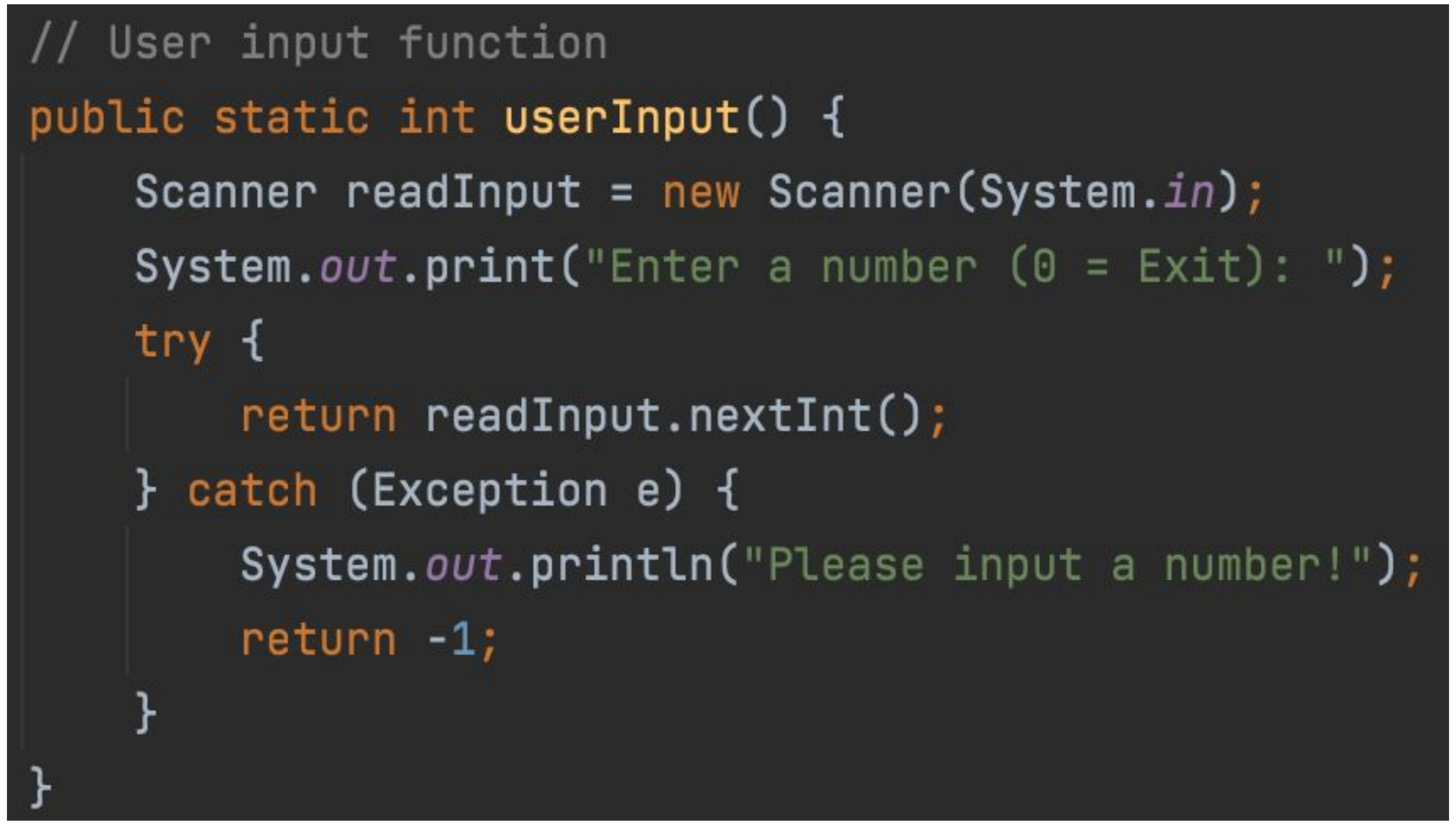
```
+---------------------------+
|      Vending Machine      |
+---------------------------+
|   1     water €1.0 [10]   |
|   2    sprite €1.2 [10]   |
|   3      cola €1.3 [10]   |
|   4     fanta €1.1 [10]   |
|   5    orange €1.5 [10]   |
+---------------------------+
Enter a number (0 = Exit):
```



```
Enter a number (0 = Exit): 0
Amount to pay = 3,5
```

Exercise 1: Create a Vending Machine

Useful functions to capture user input:



```java
// User input function
public static int userInput() {
    Scanner readInput = new Scanner(System.in);
    System.out.print("Enter a number (0 = Exit): ");
    try {
        return readInput.nextInt();
    } catch (Exception e) {
        System.out.println("Please input a number!");
        return -1;
    }
}
```

How are you feeling?

RED
I have no idea what you're talking about

YELLOW
I have some questions but feel like I understand some things

GREEN
I feel comfortable with everything you've said

Object Oriented Programming

OOP

OOP stands for Object Oriented Programming

- Object-oriented programming is about creating objects that contain both data and methods.

- Object-oriented programming advantages:
  - OOP is faster and easier to execute
  - OOP provides a clear structure for the programs
  - OOP helps to keep the Java code DRY "Don't Repeat Yourself", and makes the code easier to maintain, modify and debug
  - OOP makes it possible to create full reusable applications with less code and shorter development time

Classes

Classes

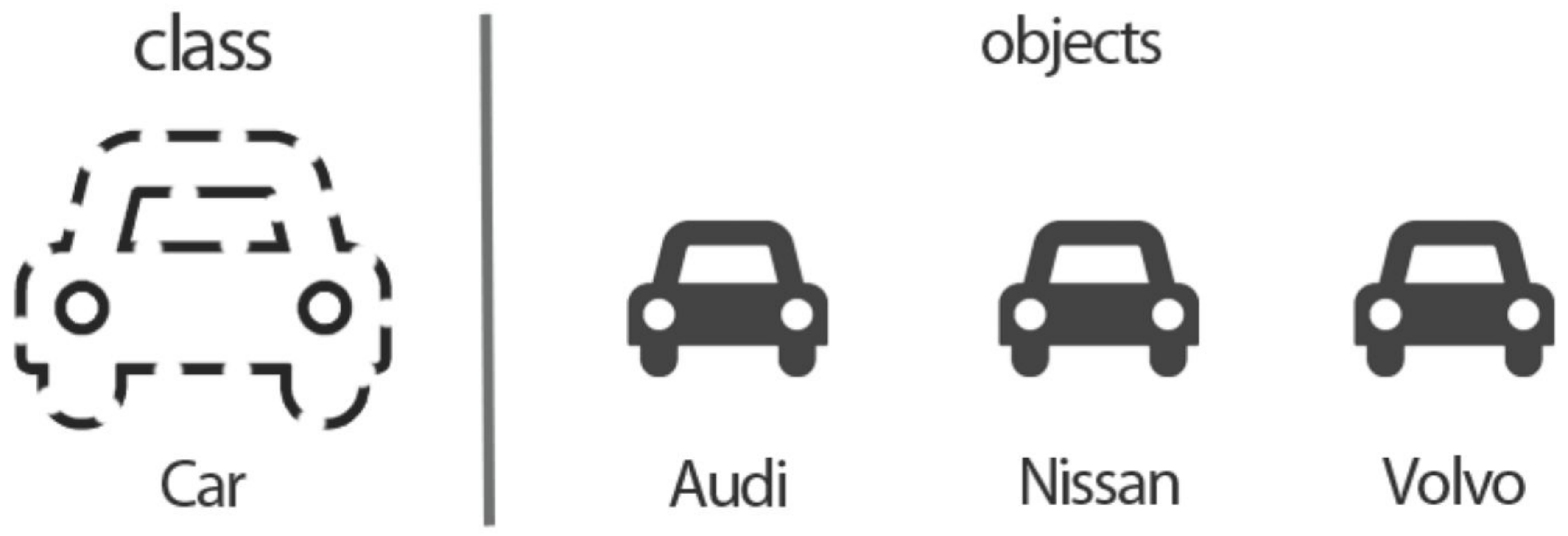
A Class is a blueprint for creating Objects

- A class is a template for objects. When the individual objects are created, they inherit all the variables and methods from the class.
- Everything in Java is associated with classes and objects, along with its attributes and methods.
- For example: in real life, a car is an object. The car has attributes, such as weight and color, and methods, such as drive and brake.

Class vs. Object

A Class is a blueprint for creating Objects

- The Car Class is a template. The CarClass **instantiated** to create Car objects. Each Car object can be slightly different ie. might have different names, colours etc but will inherit all the attributes in the Car class. Let us look at an example...



Class

In the example on the right, the Car class contains an attribute (aka variable), which is the number of wheels, and a method called honk.

So, any Car object that is created will have 4 wheels and a 'honk' method.

Class

```
class Car {
  private int Wheels = 4;         // Vehicle attribute
  public void honk() {                     // Vehicle method
    System.out.println("Tuut, tuut!");
  }
}
```

Class

In the example on the right, the **Car** class contains an attribute (aka variable) and a method.

Any Car object that is created will have 4 wheels and a 'honk' method.

Class

```
class Car {
  private int Wheels = 4;         // Vehicle attribute
  public void honk() {                      // Vehicle method
    System.out.println("Tuut, tuut!");
  Car Ford = new Car();
  }
}
```

Methods

Methods

A method is a block of code which runs when it is called

- Methods are used to perform actions, and they are also known as functions.

- A method essentially holds a block of code. Methods have names, which are used to ‘call’ or invoke them.

- To call a method in Java, write the method's name followed by two parentheses () and a semicolon;
  - The code inside the method will then be executed

Method

Creating a method:

- myMethod() is the name of the method
- static means that the method belongs to the Main class and is not an object of the Main class.
- void means that this method does not have a return value

Create a Method

```
public class Main {
  static void myMethod() {
    System.out.print(“I just made a method”);
  }
}
```

Method

Calling a method:

To call a method, write the method's name followed by two parentheses () and a semicolon;

Calling a Method

```
public static void main(String[] args) {
    myMethod();
  }
```

Method

Calling a Method

```
public static void main(String[] args) {

     myMethod();

     myMethod();

     myMethod();

   }
```

Calling a method:

A method can also be called multiple times.

Inheritance

Inheritance

What is Inheritance in Java?

- In Java, it is possible to inherit attributes and methods from one class to another.

- We can think about it in two categories:
  - subclass (child) - the class that inherits from another class
  - superclass (parent) - the class being inherited from

- To inherit from a class, use the extends keyword.

Inheritance

In the example on the right, the **Car** class (subclass) inherits the attributes and methods from the **Vehicle** class (superclass):

Inheritance

```
class Vehicle {
  protected String brand = "Ford";        // Vehicle attribute
  public void honk() {                    // Vehicle method
    System.out.println("Tuut, tuut!");
  }
}

class Car extends Vehicle {
  private String modelName = "Mustang";    // Car attribute
  public static void main(String[] args) {

    // Create a myCar object
    Car myCar = new Car();

    // Call the honk() method (from the Vehicle class) on the myCar object
    myCar.honk();

    // Display the value of the brand attribute (from the Vehicle class) and the
value of the modelName from the Car class
    System.out.println(myCar.brand + " " + myCar.modelName);
  }

}
```

How are you feeling?

RED
I have no idea what you're talking about

YELLOW
I have some questions but feel like I understand some things

GREEN
I feel comfortable with everything you've said

Abstraction

Abstraction

What is Abstraction?

- Data abstraction is the process of hiding certain details and showing only essential information to the user.
- Abstraction can be used for security: by hiding certain details and only showing the important details of an object
- We will not go into detail today but it is worth noting that Abstraction can be achieved with either abstract classes or interfaces

Encapsulation

Encapsulation

What is Encapsulation?

Encapsulation is used to make sure that "sensitive" data is hidden from users.

To achieve this, you must:

- declare class variables/attributes as private
- provide public **get** and **set** methods to access and update the value of a private variable

Encapsulation

private variables can only be accessed within the same class (an outside class has no access to it).

However, it is possible to access them if we provide public get and set methods.

The get method returns the variable value, and the set method sets the value.

Encapsulation

```
public class Person {
  private String name; // private = restricted access

  // Getter
  public String getName() {
    return name;
  }

  // Setter
  public void setName(String newName) {
    this.name = newName;
  }
}
```

How are you feeling?

RED

I have no idea what you're talking about

YELLOW

I have some questions but feel like I understand some things

GREEN

I feel comfortable with everything you've said

Exercise 2: OOP

Exercise 2: OOP

We are now going to create our own OOP programme using what we have just learnt.

Task: We want to store information about authors, books, and be able to retrieve this information.

1) Creating the Author class:

   Will any variables be private?

   What will our constructor do?

   What getters and setters will we need?

Author Class

In breakout groups, code the following class we designed for Author.

Create a class called BookShopTest and create an Author object using the Class.

```
Author
-----------------------------------------------
name: String
address: String
phonenumber: Int
-----------------------------------------------

+ Author(name:String, address:String,
phonenumber: Int)

+getName(): String

+getAddress(): String

+ setAddress(address: String): void

+ getPhoneNumber(): Int

+toString(): String
```

Book Class

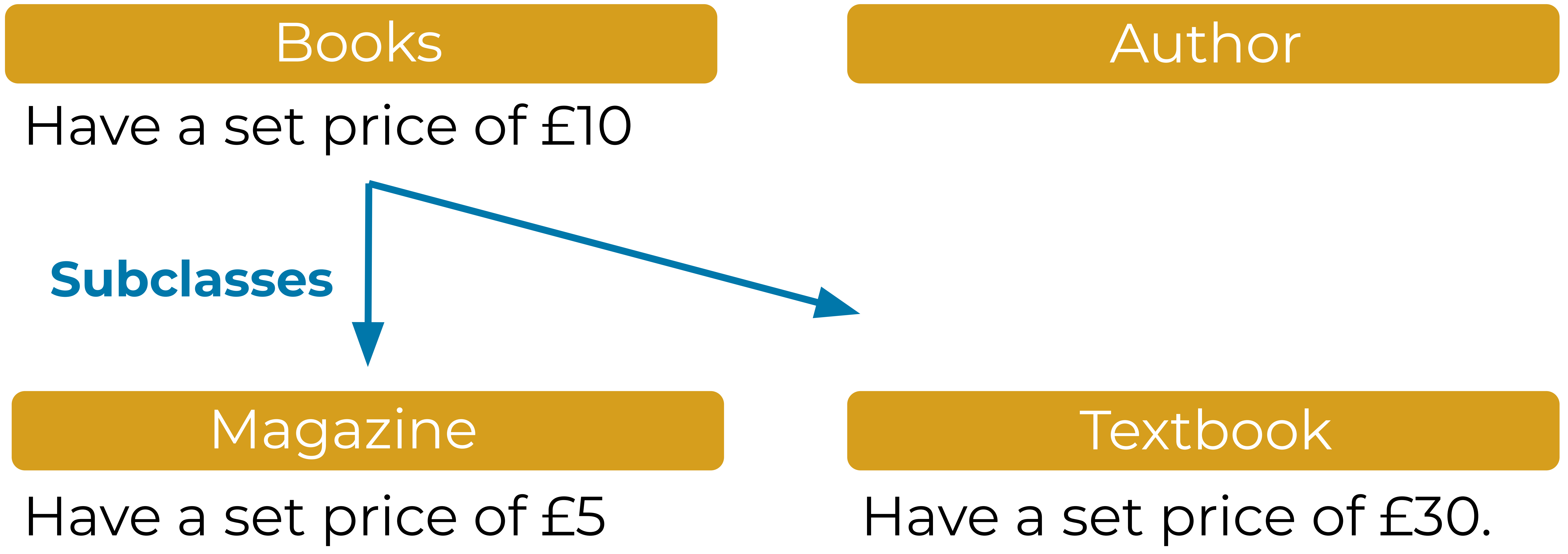
2) Create your book class.

Think about the class diagram/design.

Book
?
?

Magazines and Textbooks

3) Magazine and Textbooks will inherit from the ‘Book’ class but have a different price.
Task: Create our Magazine and Textbook Subclasses



Have a set price of £10

Have a set price of £5

Have a set price of £30.