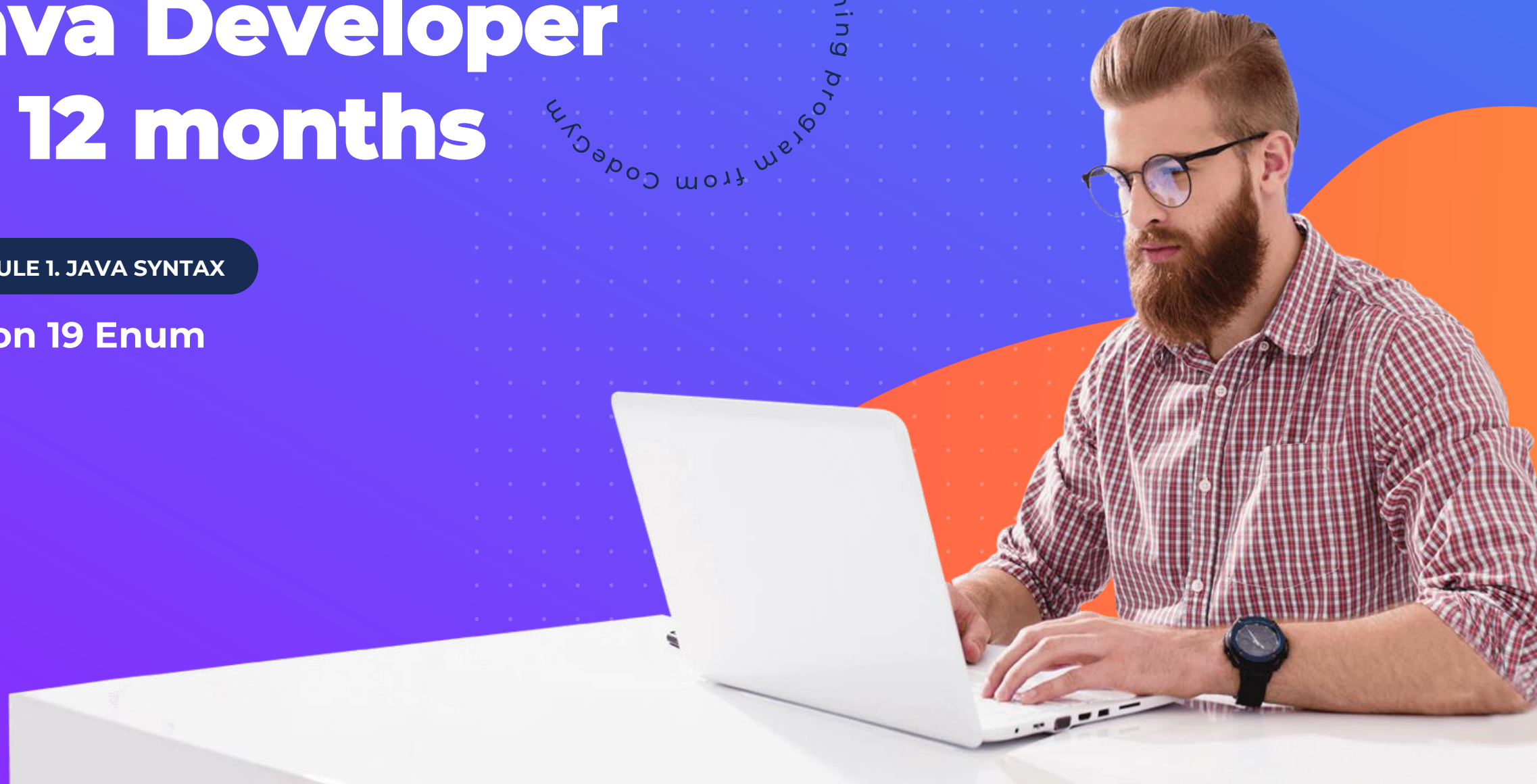
CODEGYM

Java Developer in 12 months

Mentor-supported training program from CodeGym

MODULE 1. JAVA SYNTAX

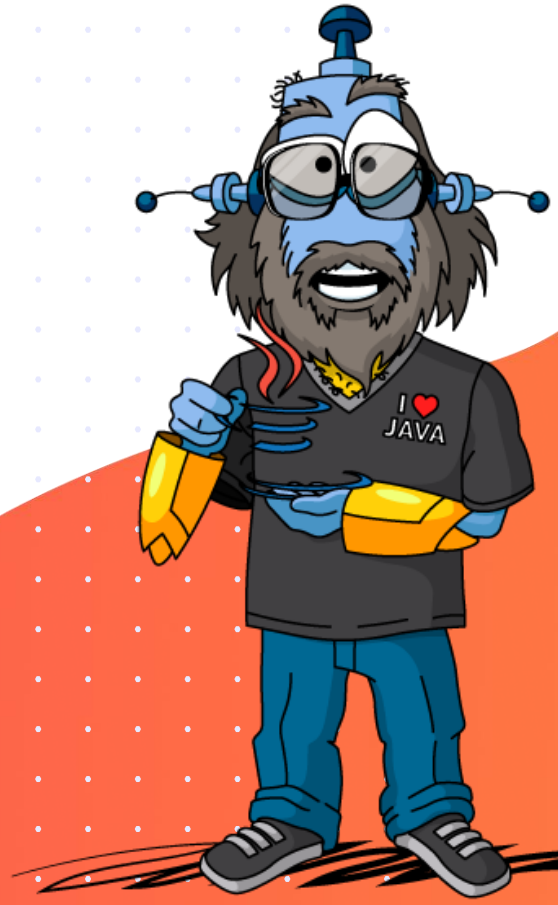
Lesson 19 Enum



CODEGYM

Lesson plan

- Enum
- Singleton
- The multiple-choice operator: switch



CODEGYM

New data type: enum or enumeration

The name enum comes from the word enumeration.

enum is a data type that consists of a finite set of values.

For example, a DayOfTheWeek type can only take the values MONDAY, TUESDAY, WEDNESDAY, ... There are 7 values in total.

Or a Month type can only take only the values JANUARY, FEBRUARY, MARCH, ... There are 12 values in total.

CODEGYM

Declaring a type

Declaring a new enum data type:

```
enum TypeName {
  VALUE1,
  VALUE2,
  VALUE3
}
```

TypeName is the name of the new type (class), and the possible values are separated by commas and wrapped in curly braces: Value1, Value2, Value3.

Here is how you assign a value to a variable of our new type:

```
Day day = Day.MONDAY;
```

Code	Note
Day day = Day.FRIDAY; System.*out*.println(day);	The screen output will be: FRIDAY

CODEGYM

Methods of an enum

The static **values()** method returns an array of all of the values of the enum type:

Code	Note

`Day[] days = Day.values();`
 	
`for (Day day: days)`
`  System.out.println(day);`
 	
`System.out.println(days[2]);`
	The days variable stores an array containing the values of the Day type (7 elements)
	
	Display the contents of the array on the screen:
	MONDAY
	
	TUESDAY
	WEDNESDAY
	THURSDAY
	FRIDAY
	SATURDAY
	SUNDAY
	
	WEDNESDAY

The **ordinal()** method returns the ordinal number of the value (constant).

You call it on an enum value rather than an enum class:

Code	Console output

`System.out.println(Day.MONDAY.ordinal());`
`System.out.println(Day.FRIDAY.ordinal());`
`System.out.println(Day.SUNDAY.ordinal());`
	0
	4
	6

CODEGYM

More methods of an enum

Converting to and from a string
To convert an enum object to a string, you need to call its **toString**() method.

```
String str = Day.MONDAY.toString();
```

To convert in the other direction (from a string to a Day object), you can use the static **valueOf()** method:

```
Day day = Day.valueOf("MONDAY");
```

Converting to a number and back again
To convert an enum object to a number, you need to call its **ordinal**() method:

```
int index = Day.MONDAY.ordinal();
```

To convert in the other direction (from a number to a Day object), you need a more transparent construct:

```
Day day = Day.values()[2];
```

Important point: because enum values are a fixed set of constants, **they can be compared using** ==. In other words, you can't have two identical MONDAY objects with different address. Only a single instance of each enum value exists. And that means that comparing enum variables using == will always work.

CODEGYM

Converting to a class

In reality, there's nothing magical here. The compiler just gave us some syntactic sugar. At compile time, the Day enum is converted to an ordinary class:

```
public class Day {
   public static final Day MONDAY = new Day(0);
   public static final Day TUESDAY = new Day(1);
   public static final Day WEDNESDAY = new Day(2);
   public static final Day THURSDAY = new Day(3);
   public static final Day FRIDAY = new Day(4);
   public static final Day SATURDAY = new Day(5);
   public static final Day SUNDAY = new Day(6);

   private static final Day[] array = {MONDAY, TUESDAY,
         WEDNESDAY, THURSDAY, FRIDAY, SATURDAY, SUNDAY};

   private final int value;

   private Day(int value) {
      this.value = value;
   }

   public int ordinal() {
      return this.value;
   }

   public static Day[] values() {
      return array;
   }
}
```

Day class
List of static constant values

Array with all values of the Day type

Variable that stores the value of a specific Day object

private Day class constructor — Day objects can only be created inside the Day class.

The ordinal method must be called on a Day object.

It returns the object's value — the value field.
The method returns a static array with all the values of the Day class

CODEGYM

Adding your own methods to an enum

Because an enum turns into an ordinary class at compile time, you can declare methods in it. These methods are simply added to the class that the compiler generates.

Code	Note

enum Day {
  MONDAY,
  TUESDAY,
  WEDNESDAY,
  THURSDAY,
  FRIDAY,
  SATURDAY,
  SUNDAY;
	A semicolon is required after the list of values.

public static List<Day> asList() { ArrayList<Day> list = new ArrayList<Day>();	Create an ArrayList object.
Collections.*addAll*(list, *values*());	Add the values in the array returned by the values()
return list; } }	Return the list.

CODEGYM

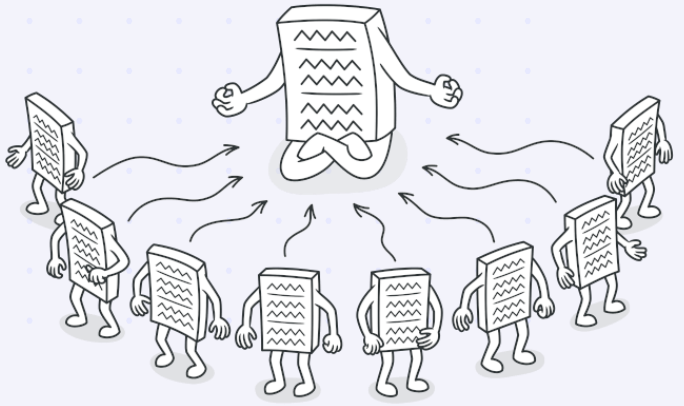
Singleton

Singleton is one of the simplest class-level design patterns. Sometimes people say "this class is singleton", which means that the class implements the singleton design pattern.

Sometimes it is necessary to write a class where we restrict instantiation to a single object. For example, a class responsible for logging or connecting to a database.

A singleton is a design pattern that does two things:

- **It guarantees that there will only ever be one instance of the class.**
- **It provides a single point of global access to that instance.**



CODEGYM

A simple singleton implementation

There are two features that are characteristic of nearly every implementation of the singleton pattern:

1. The constructor is private. This limits the ability to create objects of the class outside of the class itself.
2. A public static method that returns the instance of the class. This method is called getInstance. This is the point of global access to the class instance.

```
public class Singleton {
   private static final Singleton INSTANCE = new Singleton();

   private Singleton() {
   }

   public static Singleton getInstance() {
      return INSTANCE;
   }
}
```

CODEGYM

Java's multiple-choice operator: switch

Java has another interesting operator. We're talking about the switch statement. We could also call it a multiple-choice operator. It looks like this:

```
switch(expression) {
  case value1: code1;
  case value2: code2;
  case value3: code3;
}
```

An expression or variable is indicated within the parentheses. If the value of the expression is value1, the Java machine starts executing code1. If the expression is equal to value2, execution jumps to code2. If the expression is equal to value3, then code3 is executed.

CODEGYM

Break statement in switch

An important feature of a switch statement is that the program simply jumps to the required line (to the required code block) and then executes all the blocks of code until the end of the switch.

Not only the block of code corresponding to the value in the switch, but all the blocks of code until the end of the switch.

If you want to execute only one block of code — the block of code associated with the matched case — then you need to end the block with a break statement;

```
switch(expression) {
  case value1: code1;break;
  case value2: code2;break;
  case value3: code3;
}
```

You can omit the break in the last case of the switch statement,
since that block is the last with or without a break statement.

CODEGYM

Default action: default

What happens if none of the cases listed in the switch match the expression in the parentheses?

If a matching case is not found, then the rest of the switch statement is skipped, and the program will continue execution after the curly brace ending the switch statement.

That said, you can also make a switch statement behave like the else branch in an if-else statement. To do this, use the default keyword.

```
switch(expression) {
  case value1: code1;break;
  case value2: code2;break;
  default: defaultCode;
}
```

CODEGYM

What expressions can be used in a switch statement?

Not all types can be used as case labels in a switch statement.

You can use literals of the following types:

- integer types: byte, short, int
- char type
- String type
- any enum type

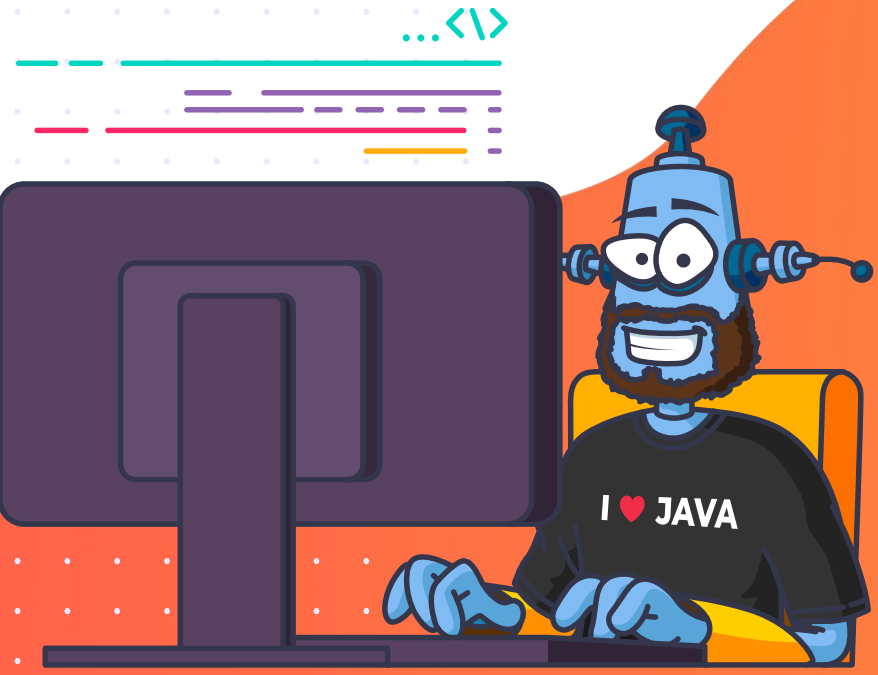
You cannot use any other types as case labels.

CODEGYM

Homework

MODULE 1. JAVA SYNTAX

Complete Level 20



CODEGYM

Answers to questions

MODULE 1. JAVA SYNTAX