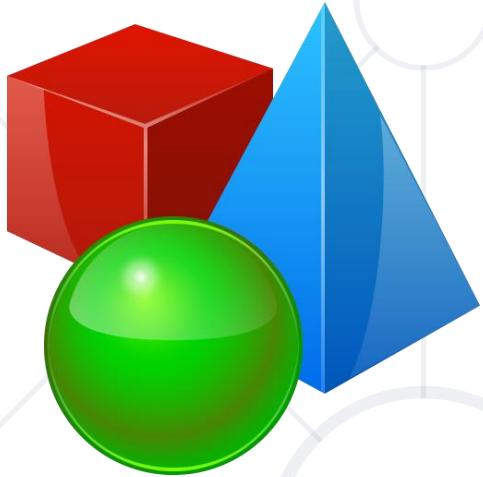# Defining Classes

Classes, Fields, Constructors, Methods

SoftUni Team

Technical Trainers

Software University

SoftUni

sli.do

#java-advanced

Table of Contents

1. Defining Simple **Classes**

2. **Fields**

3. **Methods**

4. **Constructors**, Keyword **this**

5. **Static** Members

Defining Classes

Defining Simple Classes

- Specification of a given type of object from the real-world

- **Classes** provide structure for describing and creating objects

Class **name**

Keyword

```
class Car {

   …

}
```

Class **body**

Naming Classes

- Use PascalCase naming

- Use descriptive nouns

- Avoid ambiguous names

```
class Dice { … }

class BankAccount { … }

class IntegerCalculator { … }
```

```
class TPMF { … }

class bankaccount { … }

class numcalc { … }
```

Class Members

- Class is made up of **state** and **behavior**

- Fields **store state**

- Methods **describe behaviour**

```
class Car {
    String brand;          Fields
    String model;
    void start(){ … }      Method
}
```

```
class Dog {
    int age;               Fields
    String type;
    void bark(){ … }       Method
}
```

Creating an Object

■ A class can have **many instances** (objects)

```
class Program {

  public static void main() {

    Car firstCar = new Car();

    Car secondCar = new Car();
  }
}
```

Variable stores a reference
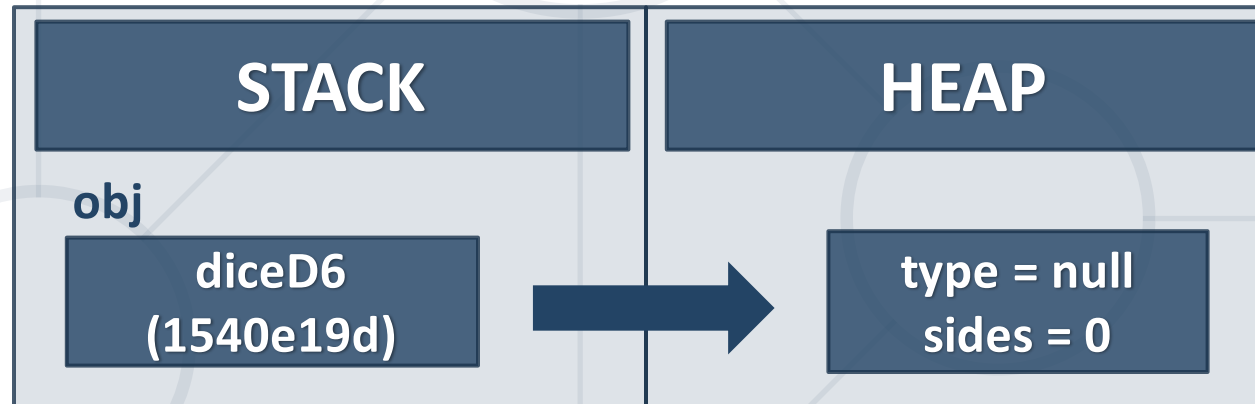
Use the new keyword

Object Reference

- Declaring a variable creates a **reference** in the stack
- The **new** keyword allocates memory on the heap

```
Car car = new Car();
```

STACK	HEAP
obj **diceD6** **(1540e19d)** →	**type = null** **sides = 0**

Classes vs. Objects

- Classes provide structure for creating objects

```
class
Dice
```
Class name

```
sides: int
```
Class fields

```
roll(…)
```
Class actions (methods)

- An object is a single instance of a class
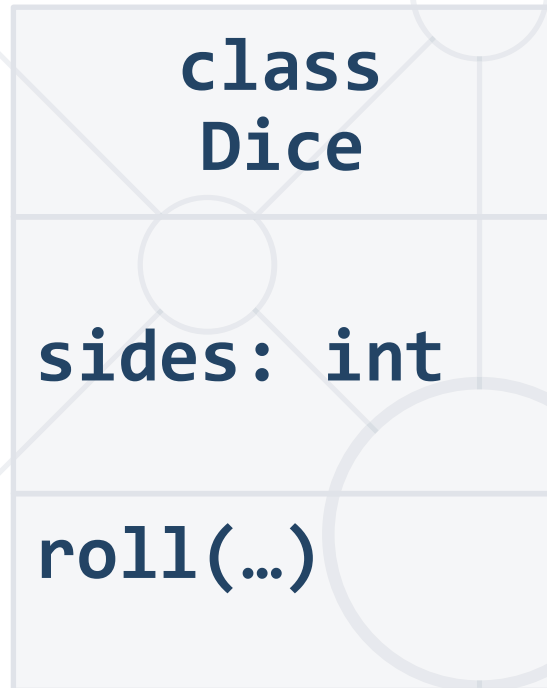
```
object
D6Dice
```
Object name

```
sides = 6
```
Object data
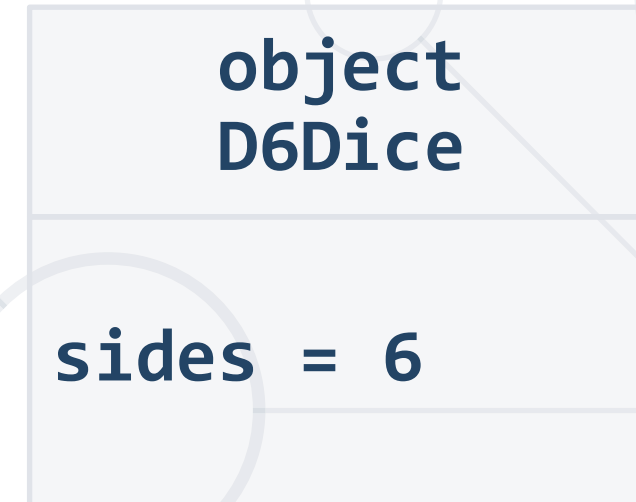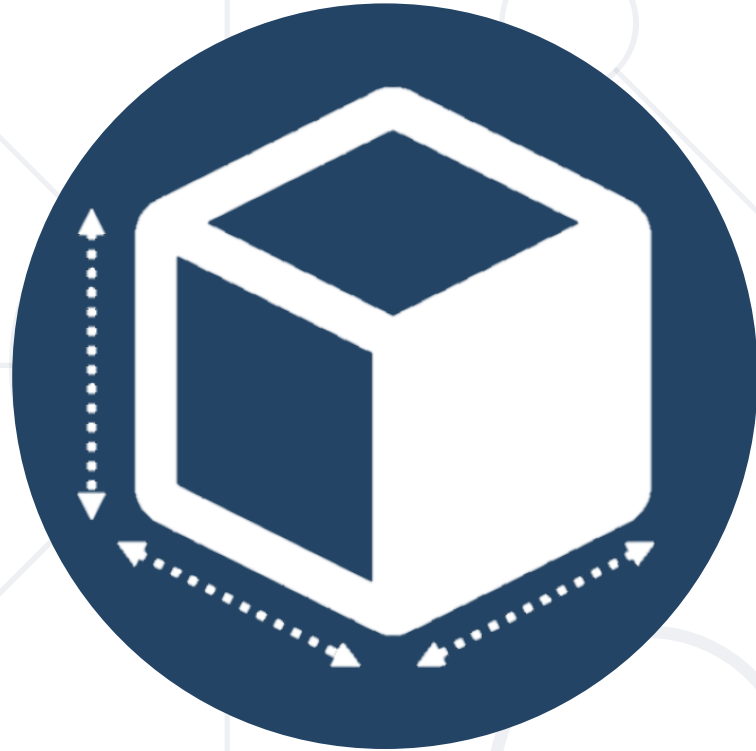
Class Data

Fields

- Class fields have **access modifiers**, **type**, and **name**

```
public class Car {

    private String brand;

    private int year;

    public Person owner;

    …

}
```

type

access modifier

name

Fields can be of any type

- Create a class **Car**

Car	Class name
+brand:String **+model:String** **+horsePower:int**	Class fields
(no actions)	Class methods
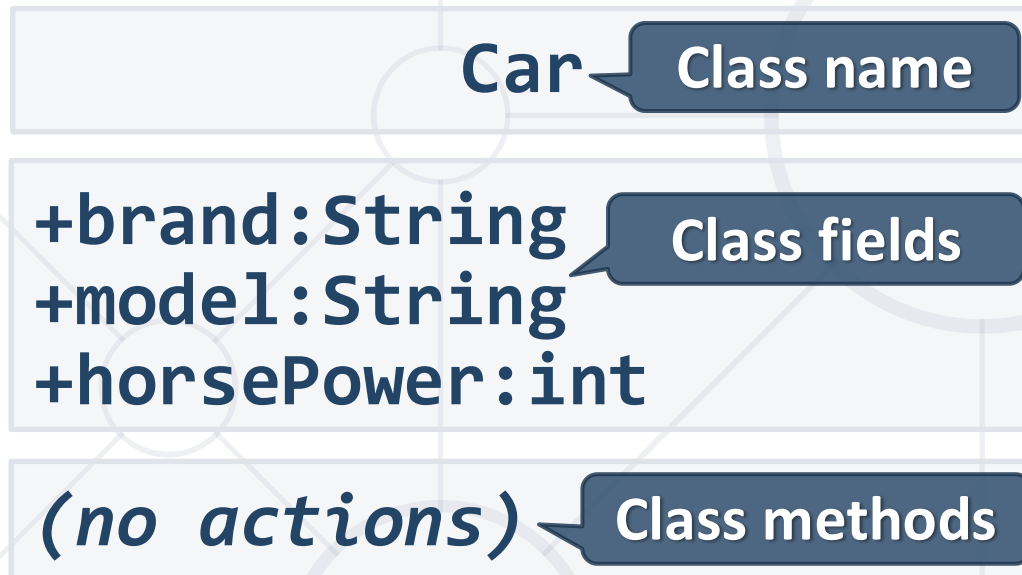
```java
public class Main {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        Car car = new Car();

        car.brand = "Chevrolet";
        car.model = "Impala";
        car.horsePower = 390;

        System.out.println(String.format(
                "The car is: %s %s - %d HP",
                car.brand, car.model, car.horsePower
        ));
    }
}
```

- Ensure proper naming!

13

```java
public class Car {
    String brand;
    String model;
    int horsePower;
}
```

Access Modifiers

- Classes and class members **have modifiers**

- Modifiers **define visibility**

Class modifier

```
public class Car {
    private String brand;
    private String model;
}
```

Member modifier

Fields should always be private!

Methods

Methods

- Store **executable code** (algorithm) that manipulate state

```
class Car {
  private int horsePower;

  public void increaseHP(int value) {
    horsePower += value;
  }
}
```

Getters and Setters

- Used to create **accessors** and **mutators** (**getters** and **setters**)

```
class Car {
    private int horsePower;
    public int getHorsePower() {
        return this.horsePower;
    }

    public void setHorsePower(int horsePower) {
        this.horsePower = horsePower;
    }
}
```

Field is hidden

Getter provides access to field

this points to the current instance

Setter provide field change

Getters and Setters

- Keyword **this**

  - Prevent field hiding

  - Refers to a current object

```
private int horsePower;
public void setSides(int horsePower) {
    this.horsePower = horsePower;
}

public void setSidesNotWorking(int horsePower) {
    horsePower = horsePower;
}
```

ToString() Method

- We can use the **toString()** method to get **String** representation of an object

- Whenever we try to print the Object reference then internally the **toString()** method is invoked

- If we did not define the toString() method in your class then Object class toString()

```
Car car = new Car();
System.out.println(car); //Car@3feba861
```

ToString() Method

- If you define the toString() method in your class then your implemented/Overridden toString() method will be called..

```java
public class Car {
…
    @Override
    public String toString() {
        return this.brand + ":" + this.model;
    }
}
```

```java
Car car = new Car();
System.out.println(car); //BMW:M3
```

Equals() Method

- In java **equals()** method is used to compare equality of two Objects

```
Car firstCar = new Car("BMW","M3");
Car secondCar = new Car("Mercedes","C63 AMG");

boolean isEqual = firstCar.equals(secondCar);

System.out.println(isEqual); //false
```

HashCode() Method

- The method returns the **hash code** for the Method class object

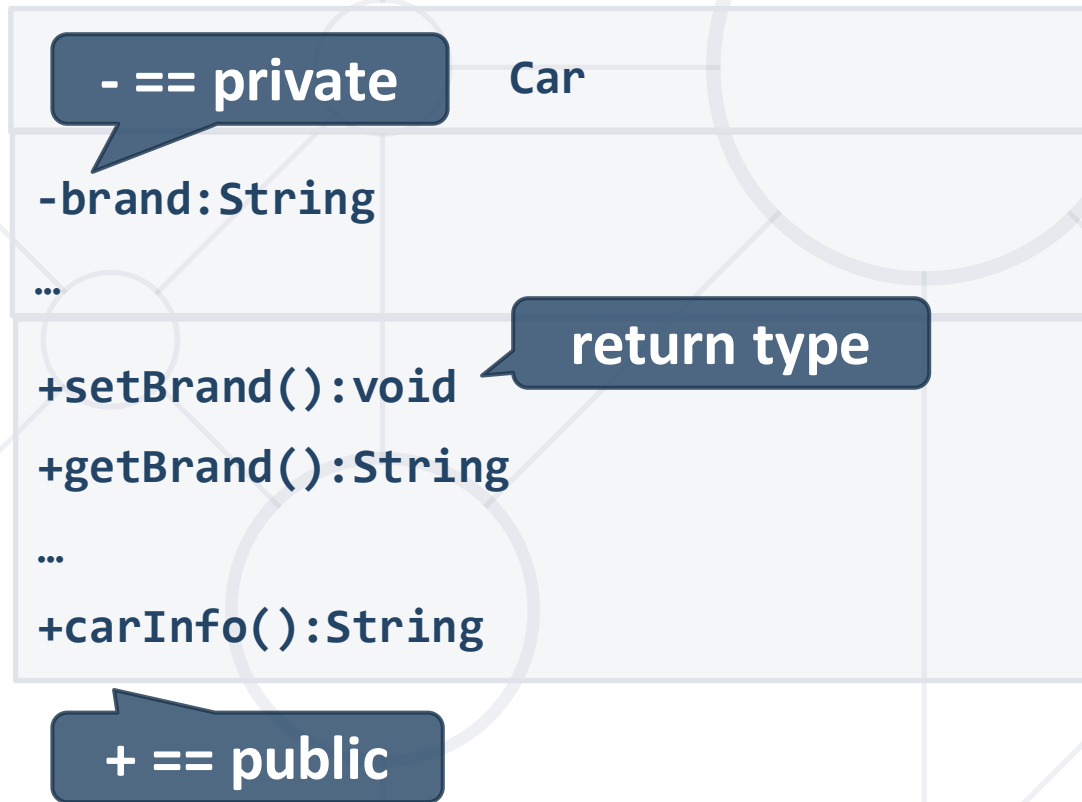- The **hash code** is always the same if the object doesn't change

  - Syntax:

```java
Car car = new Car();

int hash = car.hashCode(); //integer value which
represents hashCode value for this class.

System.out.println(hash); //1072408673
```

- Create a class **Car**

```
          Car
-brand:String
…
+setBrand():void
+getBrand():String
…
+carInfo():String
```

- == private

return type

+ == public

```java
public class Main {
    public static void main(String[] args) {
        Car car = new Car();

        car.setBrand("TESLA");
        car.setModel("MODEL S");
        car.setHorsePower(503);

        System.out.println(car.carInfo());
    }
}
```

Check your solution here: https://judge.softuni.bg/Contests/1517/Defining-Classes-Lab

Solution: Car Info

```java
public class Car {
    private String brand;
    private String model;
    private int horsePower;

    public void setBrand(String brand) { this.brand = brand; }
    public String getBrand() { return this.brand; }
    public String carInfo() {
        return String.format("The car is: %s %s - %d HP.",
        this.brand, this.model, this.horsePower);
    }
    //TODO: Create the other Getters and Setters
}
//TODO: Test the program
```

Check your solution here: https://judge.softuni.bg/Contests/1517/Defining-Classes-Lab

Constructors

Constructors

- Special methods, executed during object creation

- The only way to **call a constructor** in Java is through the **keyword new**

```java
public class Car {

    private String brand;

    public Car() {

        this.brand = "BMW";

    }

}
```

Overloading default constructor

Constructors

- Special methods, executed during object creation

```java
class Car {
  private String brand;

  …
  public Car() {
    this.brand = "unknown";

    …
  }
}
```

Overloading default constructor

Constructors

- You can have multiple constructors in the same class

```java
public class Car {
    private int horsePower; private String brand;

    public Car(String brand) {
        this.brand = brand;
    }

    public Car(String brand, int horsePower) {
        this.brand = brand;
        this.horsePower = horsePower;
    }
}
```

Constructor with one parameter

Constructor with all parameters

Object Initial State

- Constructors **set object's initial state**

```java
public class Car {
    String brand;
    List<Part> parts;

    public Car(String brand) {
        this.brand = brand;
        this.parts = new ArrayList<>();
    }
}
```

Always ensure correct state

Constructor Chaining
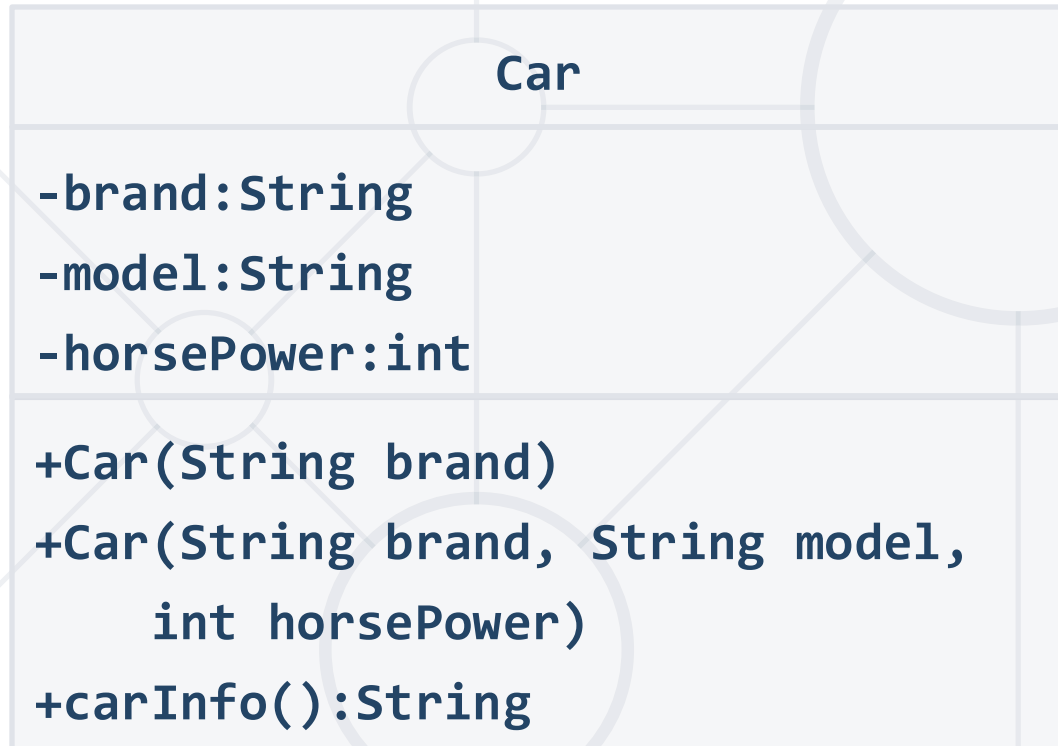
- Constructors can **call each other**

```java
class Car {
  private String brand;
  private int horsePower;

  public Car(String brand, int horsePower) {
    this.brand = brand;
    this.horsePower = horsePower;
  }

  public Car(String brand) {
    this(brand, -1);
  }
}
```

- Create a class **Car**

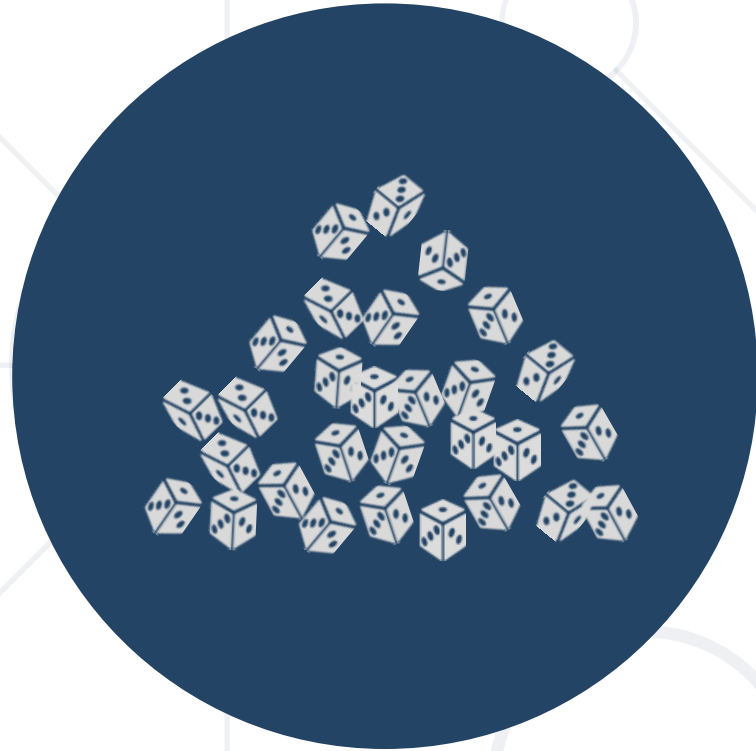| Car |
| --- |
| -brand:String
-model:String
-horsePower:int |
| +Car(String brand)
+Car(String brand, String model,
    int horsePower)
+carInfo():String |

```java
Car firstCar =
        new Car( brand: "Chevrolet");
Car secondCar =
        new Car( brand: "Chevrolet", model: "Impala", horsePower: 390);

System.out.println(firstCar.carInfo());
System.out.println(secondCar.carInfo());
```

Check your solution here: https://judge.softuni.bg/Contests/1517/Defining-Classes-Lab

Solution: Constructors

```java
public Car(String brand) {
    this.brand = brand;
    this.model = "unknown";
    this.horsePower = -1;
}

public Car(String brand, String model, int horsePower) {
    this(brand);
    this.model = model;
    this.horsePower = horsePower;
}
```

Check your solution here: https://judge.softuni.bg/Contests/1517/Defining-Classes-Lab

Static Members

Static Members

- Access static members **through the class name**

- Static members are **shared class-wide**

- You don't **need** an instance

```
class Program {

  public static void main(String[] args) {

    BankAccount.setInterestRate(2.2);

  }
}
```

Sets the rate for all bank accounts

Static Members

```java
class BankAccount {
  private static int accountsCount;
  private static double interestRate;

  public BankAccount() {
    accountsCount++;
  }


  public static void setInterestRate(double rate) {
    interestRate = rate;
  }
}
```

Problem: Bank Account

- Create a class **BankAccount**

- Support **commands**:

  - **Create**

  - **Deposit** {ID} {Amount}

  - **SetInterest** {Interest}

  - **GetInterest** {ID} {Years}

  - **End**

BankAccount

```
-id:int (starts from 1)
-balance:double
-interestRate:double (default: 0.02)
```

underline == static

```
+setInterest(double interest):void
+getId():int
+getInterest(int years):double
+deposit(double amount):void
```

```
Create
Deposit 1 20
GetInterest 1 10
End
```

➡

```
Account ID1 Created
Deposited 20 to ID1
4.00
```

(20 * 0.02) * 10

Solution: Bank Account

```java
public class BankAccount {
    private final static double DEFAULT_INTEREST = 0.02;

    private static double rate = DEFAULT_INTEREST;
    private static int bankAccountsCount;

    private int id;
    private double balance;

    // continue…
```

Check your solution here: https://judge.softuni.bg/Contests/1517/Defining-Classes-Lab

```java
    public BankAccount() {
        this.id = ++bankAccountsCount;
    }

    public static void setInterest(double interest) {
        rate = interest;
    }

    // TODO: int getId()
    // TODO: double getInterest(int years)
    // TODO: void deposit(double amount)
    // TODO: override toString()
}
```

Check your solution here: https://judge.softuni.bg/Contests/1517/Defining-Classes-Lab

Solution: Bank Account
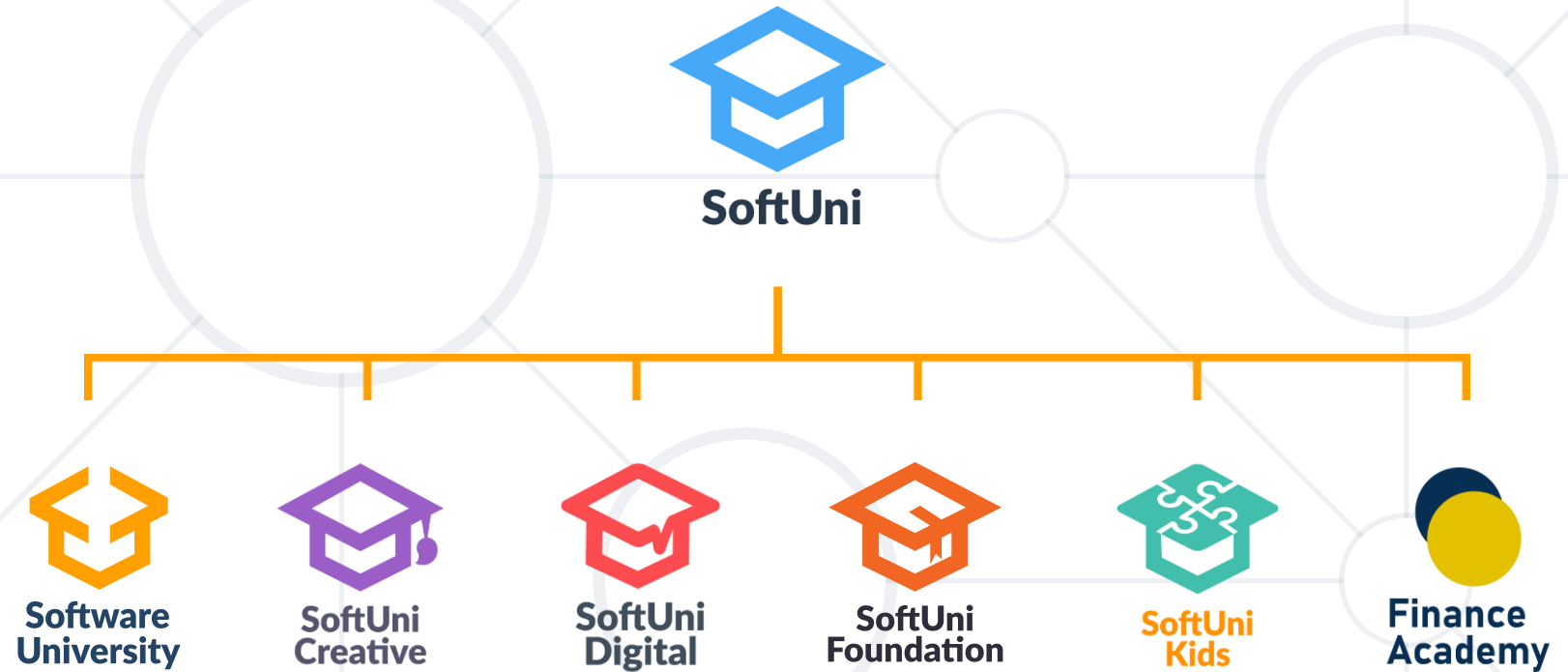
```
HashMap<Integer, BankAccount> bankAccounts = new HashMap<>();
while (!command.equals("End")) {
    //TODO: Get command args
    switch (cmdType) {
        case "Create":       // TODO
        case "Deposit":      // TODO
        case "SetInterest":  // TODO
        case "GetInterest":  // TODO
    }
    //TODO: Read command
}
```

Check your solution here: https://judge.softuni.bg/Contests/1517/Defining-Classes-Lab

Summary

- **Classes define specific structure for objects**
    - **Objects are particular instances of a class**
- **Classes define fields, methods, constructors and other members**
- **Constructors are invoked when creating new class instances**
- **Constructors initialize the object's initial state**

Questions?

SoftUni

Software University

SoftUni Creative

SoftUni Digital

SoftUni Foundation

SoftUni Kids

Finance Academy

SoftUni Diamond Partners

Trainings @ Software University (SoftUni)

- Software University – High-Quality Education, Profession and Job for Software Developers
  - softuni.bg
- Software University Foundation
  - softuni.foundation
- Software University @ Facebook
  - facebook.com/SoftwareUniversity
- Software University Forums
  - forum.softuni.bg

License

- This course (slides, examples, demos, exercises, homework, documents, videos and other assets) is **copyrighted content**

- Unauthorized copy, reproduction or use is illegal

- © SoftUni – https://about.softuni.bg

- © Software University – https://softuni.bg