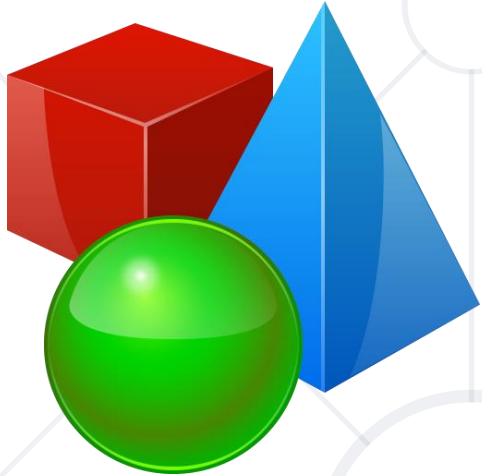


# Defining Classes

Classes, Fields, Constructors, Methods



**SoftUni Team**  
Technical Trainers



**SoftUni**



**Software University**

<https://softuni.bg>

[sli.do](https://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



Keyword



Class **name**

```
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 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

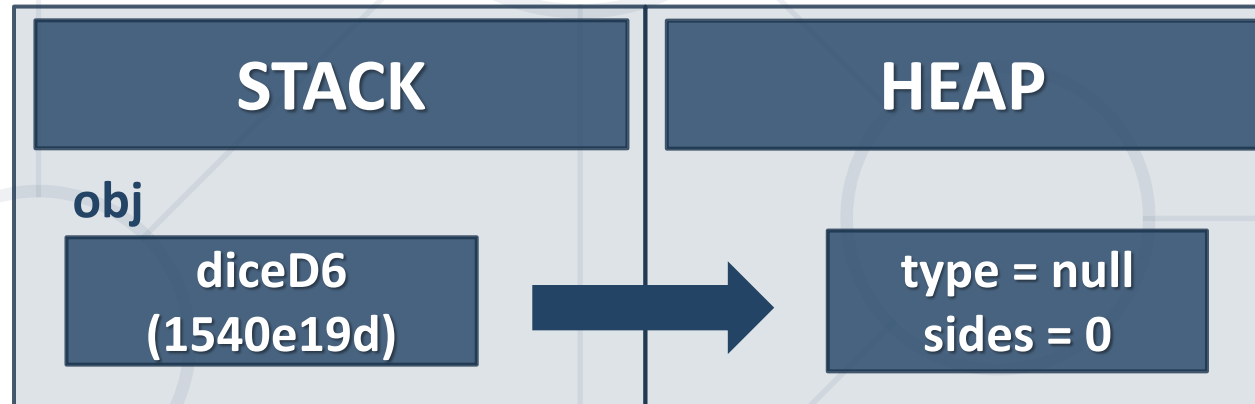




# Object Reference

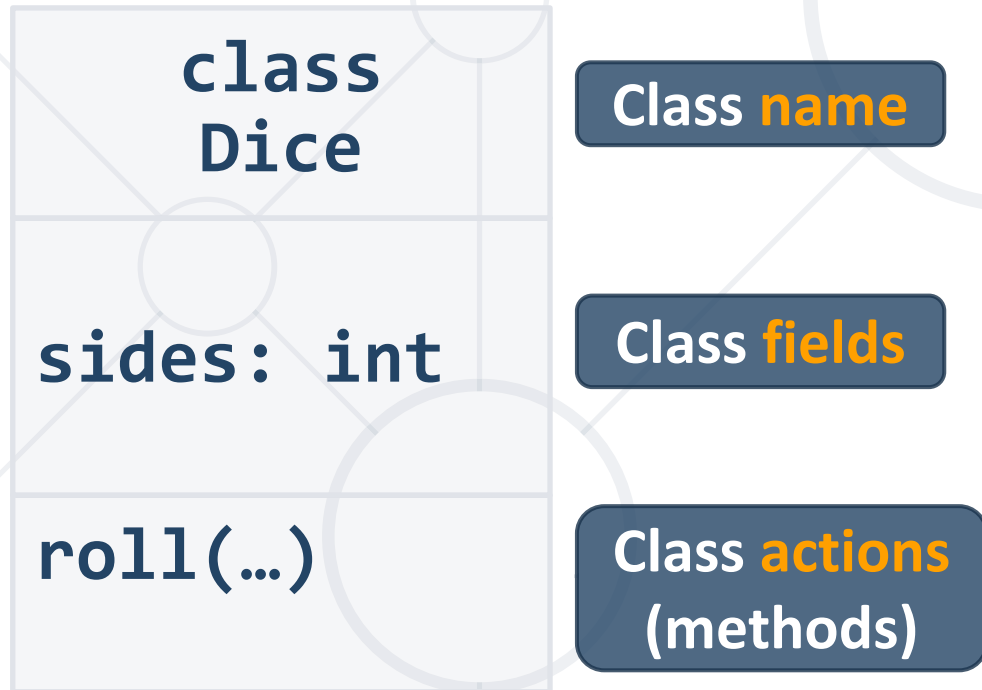
- Declaring a variable creates a **reference** in the stack
- The **new** keyword allocates memory on the heap

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

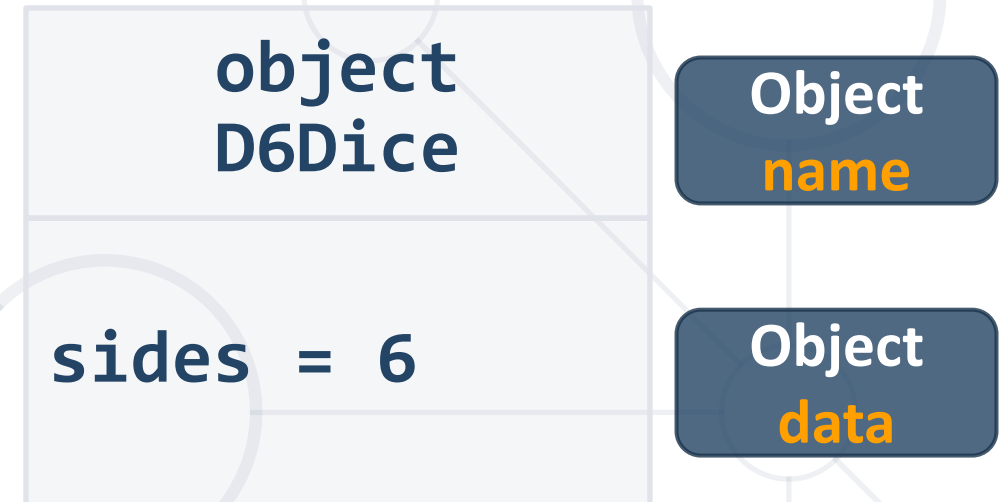


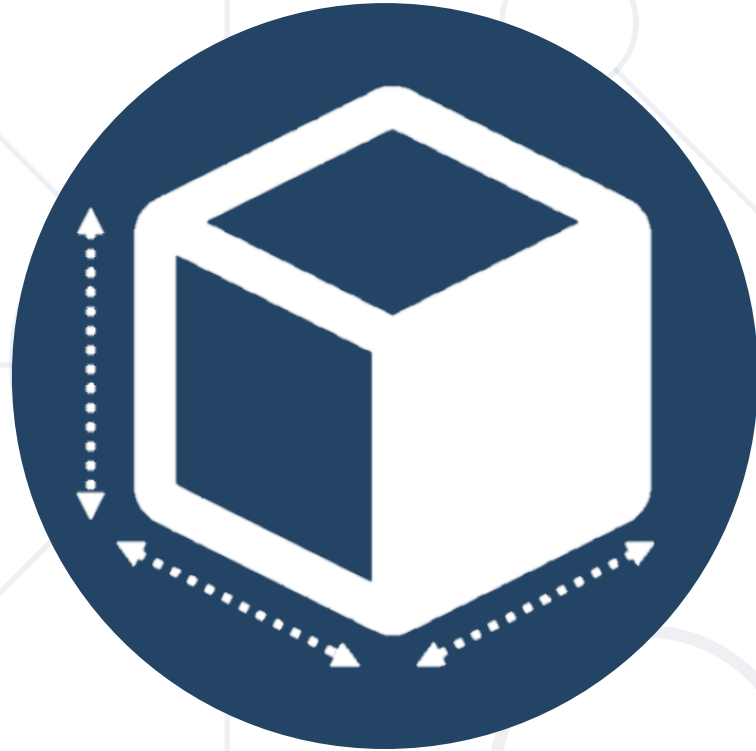
# Classes vs. Objects

- Classes provide structure for creating objects



- An object is a single instance of a class





**Class Data**

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

access modifier

```
public class Car {  
    private String brand;  
    private int year;  
    public Person owner;  
    ...  
}
```

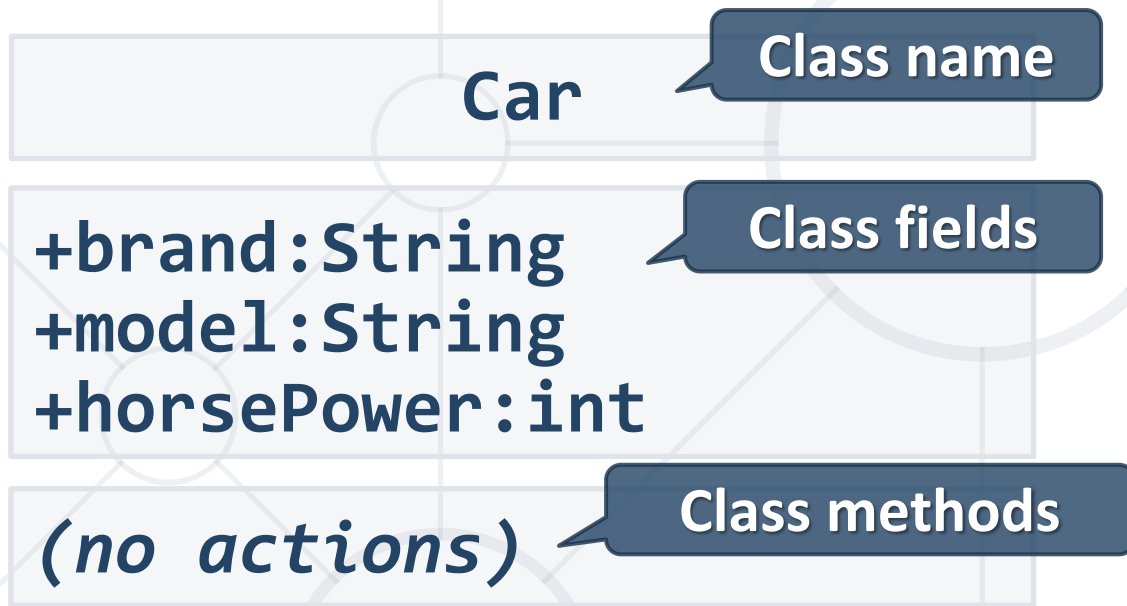
type

name

Fields can be of **any**  
**type**

# Problem: Define Car Class

- Create a class **Car**



- Ensure proper naming!

```
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  
        ));  
    }  
}
```

# Solution: Define Car Class

```
public class Car {  
    String brand;  
    String model;  
    int horsepower;  
}
```

- 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**



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

```
class Car {  
    private int horsepower;  
  
    public void increaseHP(int value) {  
        horsepower += value;  
    }  
}
```

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

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

Field is hidden

Getter provides access to field

this points to the current instance

Setter provide field change

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

- 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;  
}
```

- 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
```

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

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

```
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
```

- 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:

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

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

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

# Problem: Car Info

- Create a class **Car**

- == private

Car

-brand:String

...

+setBrand():void

+getBrand():String

...

+carInfo():String

return type

+ == public

```
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());  
    }  
}
```



# Solution: Car Info


```
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
```



**Constructors**

# Constructors

- Special methods, executed during object creation
- The only way to **call a constructor** in Java is through the **keyword new**



```
public class Car {  
    private String brand;  
    public Car() {  
        this.brand = "BMW";  
    }  
}
```

**Overloading** default  
constructor

# Constructors (1)

- Special methods, executed during object creation

```
class Car {  
    private String brand;  
    ...  
    public Car() {  
        this.brand = "unknown";  
        ...  
    }  
}
```

Overloading default  
constructor

# Constructors (2)

- You can have multiple constructors in the same class

```
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

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

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

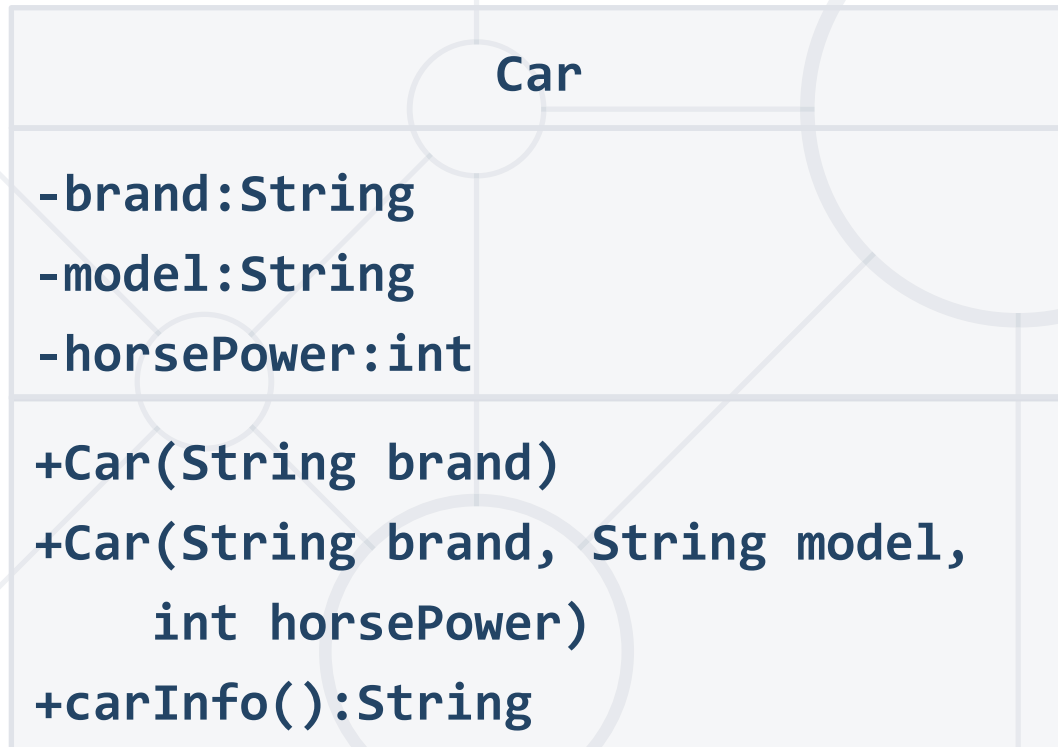
Always ensure  
correct state

- Constructors can **call each other**

```
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);  
    }  
}
```

# Problem: Constructors

- Create a class **Car**



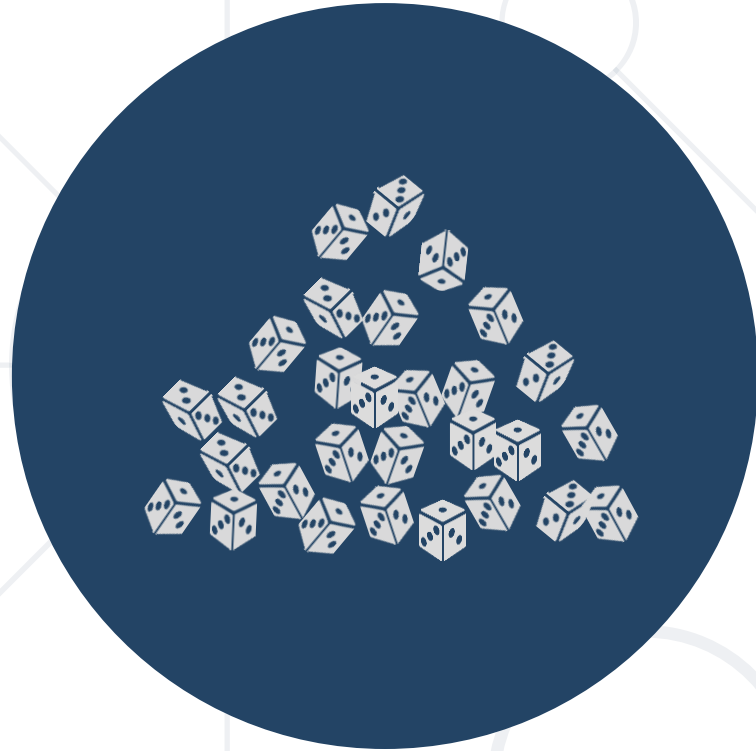
```
Car firstCar =
    new Car( brand: "Chevrolet");
Car secondCar =
    new Car( brand: "Chevrolet", model: "Impala", horsepower: 390);

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



# Solution: Constructors

```
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;  
}
```



**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

```
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)

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

**underline == static**

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



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

**$(20 * 0.02) * 10$**

# Solution: Bank Account

```
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...
```

# Solution: Bank Account (2)

```
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()  
}
```

# Solution: Bank Account (2)

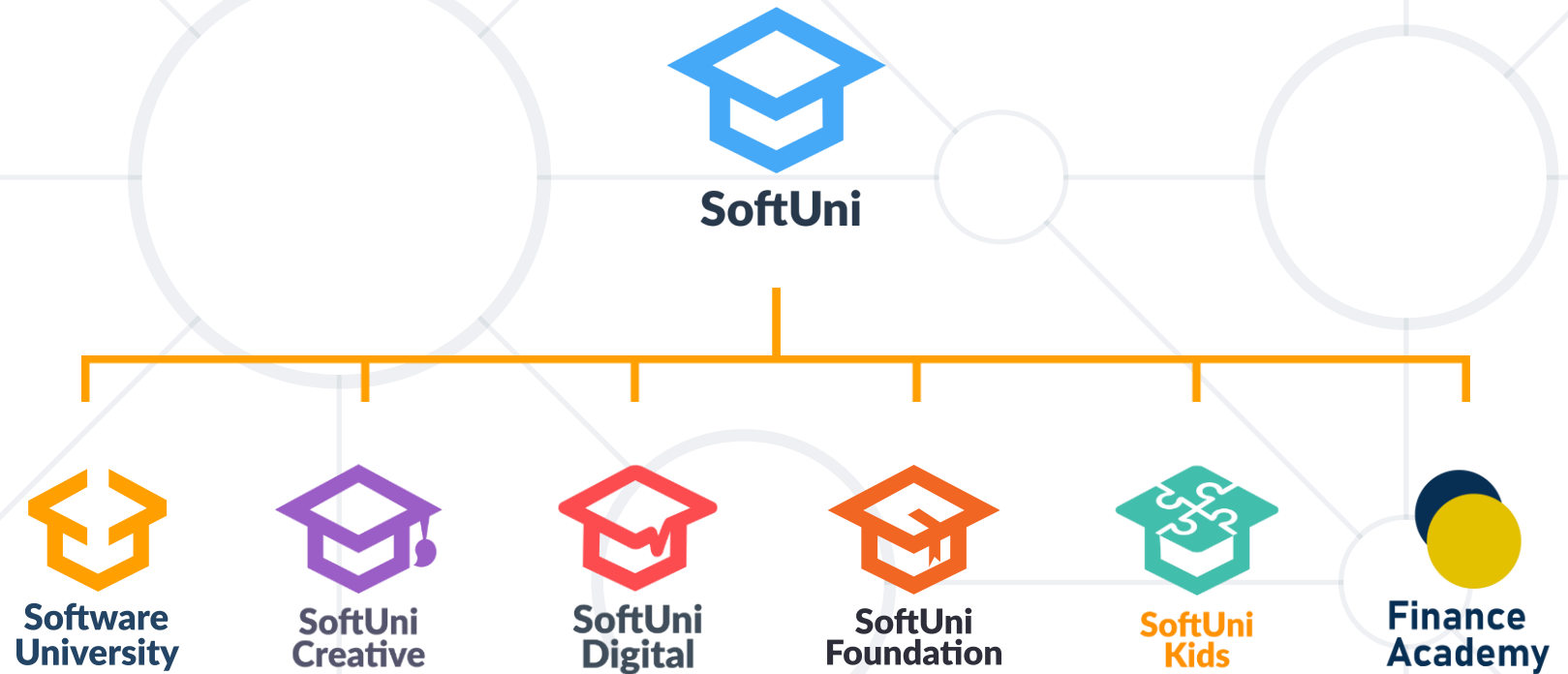
```
HashMap<Integer, BankAccount> bankAccounts = new HashMap<>();
while (!command.equals("End")) {
    //TODO: Get command args
    switch (cmdType) {
        case "Create": // TODO
        case "Deposit": // TODO
        case "SetInterest": // TODO
        case "GetInterest": // TODO
    }
    //TODO: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



- 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 Diamond Partners

**SUPER  
HOSTING  
.BG**



**Coca-Cola HBC  
Bulgaria**



**POKERSTARS**  
POKER | CASINO | SPORTS  
a Flutter International brand

**INDEAVR**  
Serving the high achievers



**AMBITIONED**

 **DRAFT  
KINGS**



**SOFTWARE  
GROUP**

createX



**Postbank**  
Решения за твоето утре

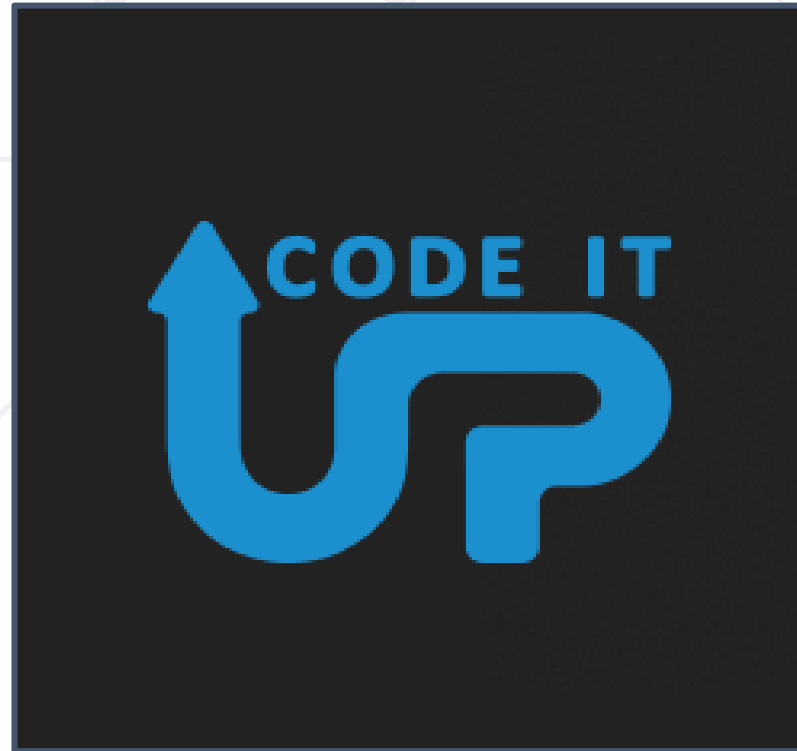


**BOSCH**

**DXC**  
TECHNOLOGY



**SmartIT**



- Software University – High-Quality Education, Profession and Job for Software Developers

- [softuni.bg](https://softuni.bg)

- Software University Foundation

- [softuni.foundation](https://softuni.foundation)

- Software University @ Facebook

- [facebook.com/SoftwareUniversity](https://facebook.com/SoftwareUniversity)

- Software University Forums

- [forum.softuni.bg](https://forum.softuni.bg)



- 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>

