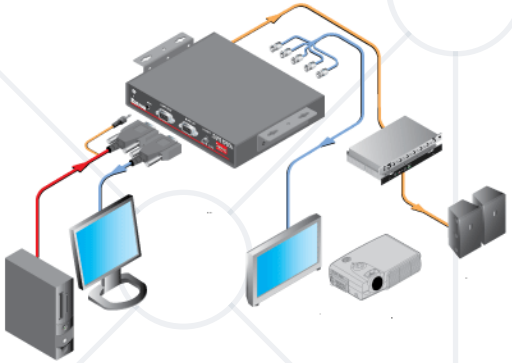
Interfaces and Abstraction

Interfaces vs Abstract Classes
Abstraction vs Encapsulation



SoftUni Team

Technical Trainers

Software University

SoftUni

Software University

https://softuni.bg

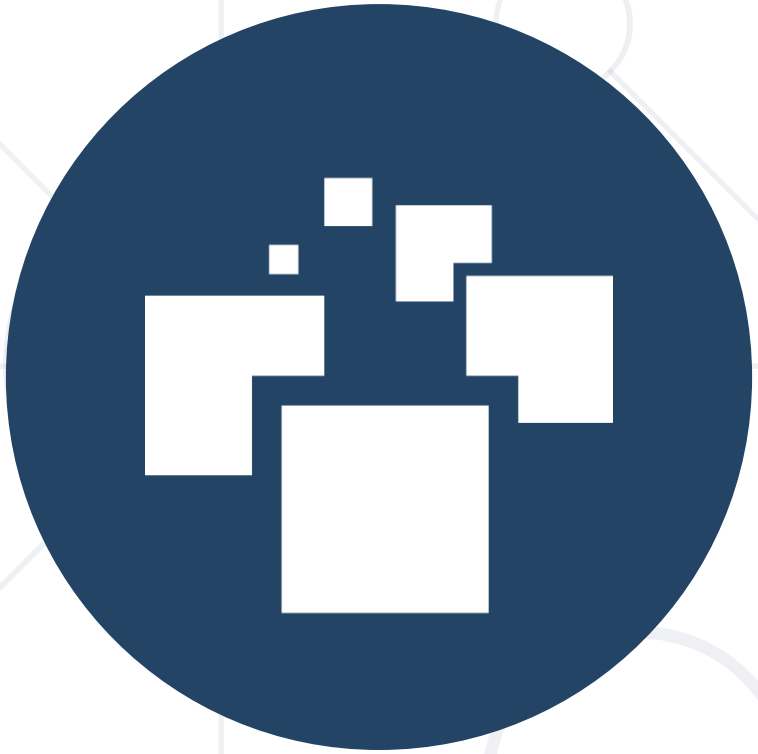
Have a Question?

Software University

sli.do

#java-advanced

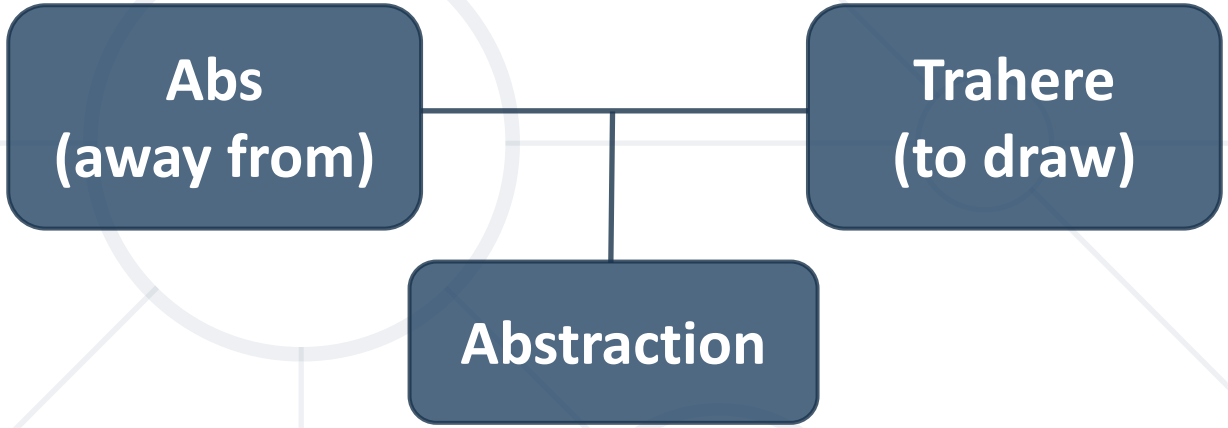
Table of Contents



Abstraction

What is Abstraction?

- Latin origin



- **Preserving** information that is **relevant** in a context
- **Forgetting** information that is **irrelevant** in that context

Abstraction in OOP

- **Abstraction** means ignoring **irrelevant** features, properties, or functions and emphasizing the **relevant ones …**

- **… relevant** to the **context** of the **project** we develop
- Abstraction helps **manage** complexity
- Abstraction lets you focus on **what the** object does instead of **how it does it**

Achieving Abstraction

- There are 2 ways to achieve abstraction in Java
  - Interfaces (**100% abstraction**)
  - Abstract class (**0% - 100% abstraction**)

```
public interface Animal {}
public abstract class Mammal {}
public class Person extends Mammal implements Animal {}
```

Abstraction vs. Encapsulation

- Abstraction
  - Process of **hiding** the **implementation details** and showing only functionality to the user
  - Achieved with **interfaces** and **abstract classes**
- Encapsulation
  - Used to **hide** the **code** and **data** inside a **single unit** to **protect** the data from the outside world
  - Achieved with **access modifiers** (private, protected, public)

Interfaces

Interface

- Internal addition by a compiler

Keyword

Public or default modifier

Name

```
public interface Printable {
   int MIN = 5;
   void print();
}
```

compiler

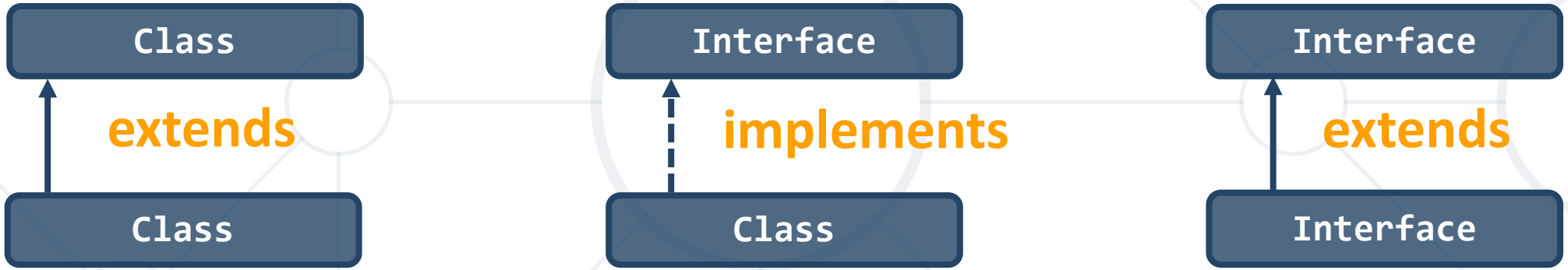
"public static final" before fields

"public" before methods

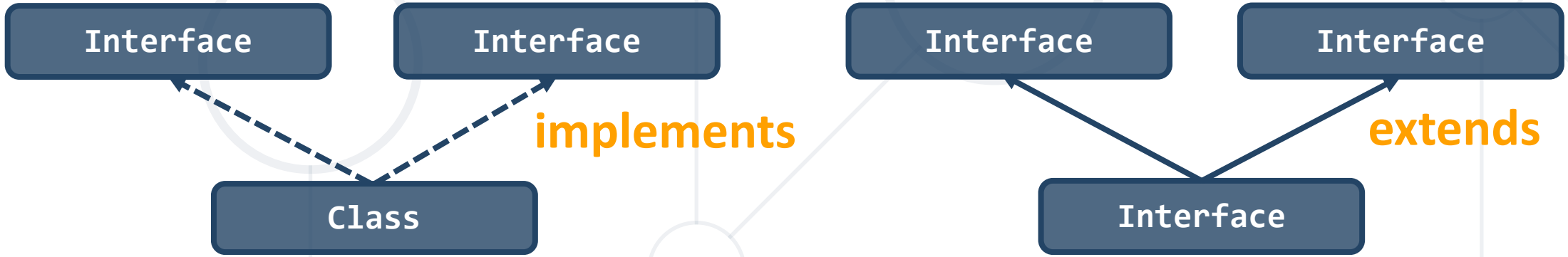
```
interface Printable {
   public static final int MIN = 5;
   public void print();
}
```

Implements vs Extends

- Relationship between classes and interfaces



- Multiple inheritances



Interface Example

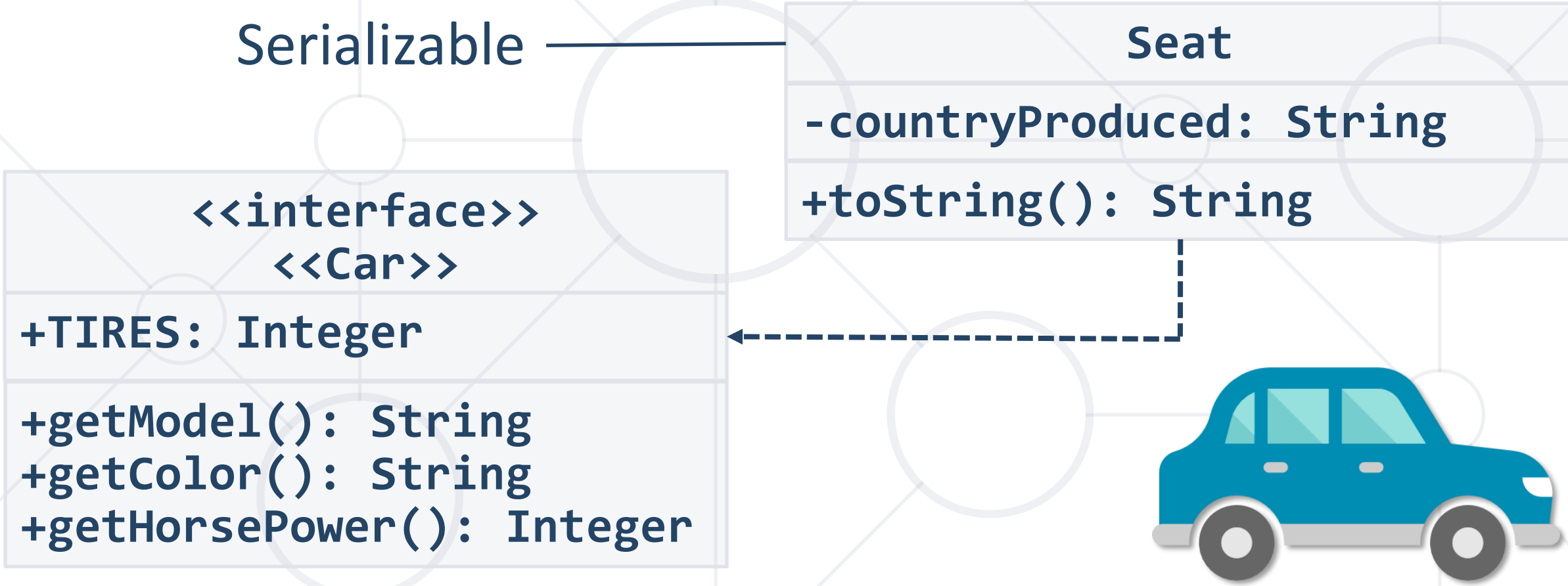
- Implementation of `print()` is provided in class `Document`

```
public interface Printable {
  void print();
}
```

```
class Document implements Printable {
  public void print() { System.out.println("Hello"); }
  public static void main(String args[]) {
    Printable doc = new Document();
    doc.print();  // Hello
  }
}
```

Polymorphism

Problem: Car Shop



Check your solution here: https://alpha.judge.softuni.org/contests/1581

Solution: Car Shop

```
public interface Car {
    int TIRES = 4;
    String getModel();
    String getColor();
    Integer getHorsePower();
    String countryProduced();
}
```

Solution: Car Shop

```
public class Seat implements Car, Serializable {
  // TODO: Add fields, constructor and private methods
  @Override
  public String getModel() { return this.model; }
  @Override
  public String getColor() { return this.color; }
  @Override
  public Integer getHorsePower() { return this.horsePower; }
}
```

Extend Interface

- The interface can **extend another interface**

```
public interface Showable  {
   void show();
}
```

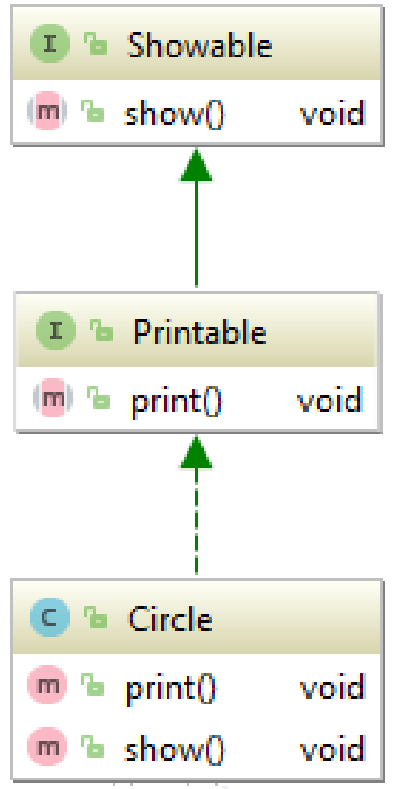
```
public interface Printable extends Showable {
   void print();
}
```

Extend Interface

- The class which implements **child** interface **must** provide an implementation for **parent** interface too

```
class Circle implements Printable
public void print() {
  System.out.println("Hello");
}

public void show() {
  System.out.println("Welcome");
}
```



Problem: Car Shop Extended

- Refactor your first problem code
  - Add interface **`Rentable`** for **rentable** cars
  - Add interface **`Sellable`** for **sellable** cars
  - Add class **`Audi`**, which **extends `CarImpl`** and **implements** rentable
  - Refactor class **`Seat`** to **extends `CarImpl`** and **implements** rentable

Solution: Car Shop Extended

```
public interface Sellable extends Car {
  Double getPrice();
}
```

```
public interface Rentable extends Car {
  Integer getMinRentDay();
  Double getPricePerDay();
}
```

Solution: Car Shop Extended

```
public class Audi extends CarImpl implements Rentable {
  public Integer getMinRentDay() {
    return this.minDaysForRent; }
  public Double getPricePerDay() {
    return this.pricePerDay; }
  // TODO: Add fields, toString() and Constructor
}
```

Default Method

- Since Java 8 we can have a **method body** in the **interface**

```
public interface Drawable {
  void draw();
  default void msg() {
    System.out.println("default method:");
  }
}
```

- If you need to **override** the default method think about your **design**

Default Method

- Implementation is **not needed** for **default methods**

```
class TestInterfaceDefault {
  public static void main(String args[]) {
    Drawable d = new Rectangle();
    d.draw();  // drawing rectangle
    d.msg();   // default method
  }
}
```

Static Method

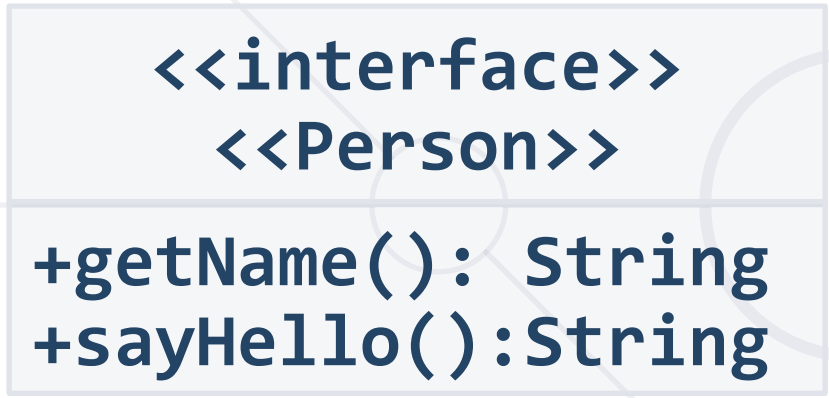
- Since Java 11, we can have a **static method** in the **interface**

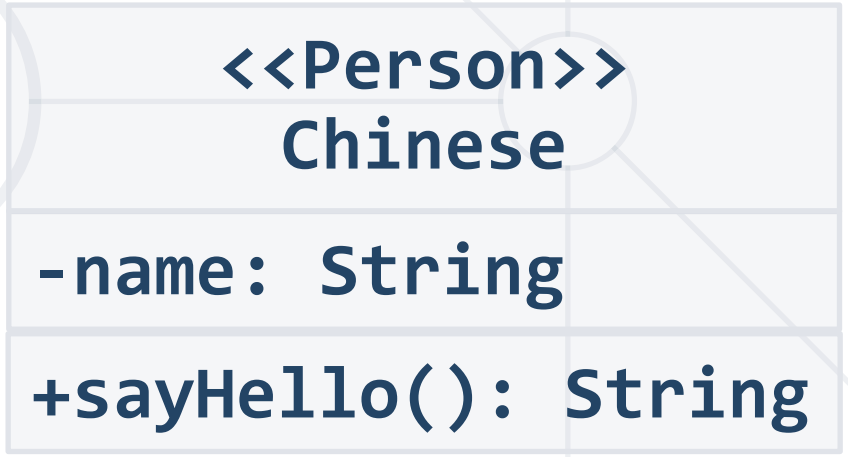
```
public interface Drawable {
   void draw();
   static int cube(int x) { return x*x*x; }
}
```

```
public static void main(String args[]) {
   Drawable d = new Rectangle();
   d.draw();
   System.out.println(Drawable.cube(3)); }  // 27
```

Problem: Say Hello

- Design a project, which has
  - **Interface** for Person
  - 3 implementations for different nationalities
  - Override where needed





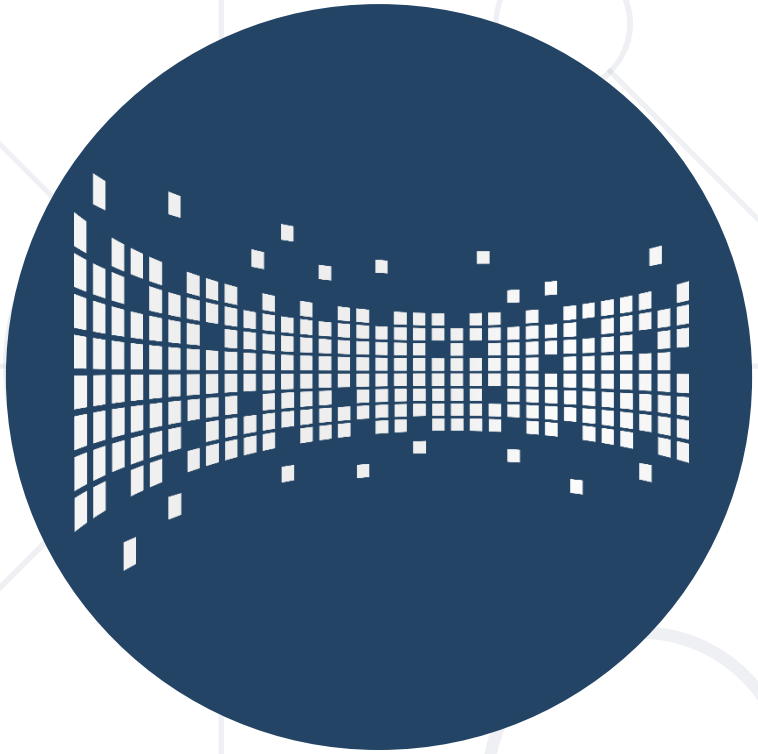
Solution: Say Hello

```
public interface Person {
  String getName();
  default String sayHello() { return "Hello"; }
}
```

```
public class European implements Person {
  private String name;
  public European(String name) { this.name = name; }
  public String getName() { return this.name; }
}
```

Solution: Say Hello

```
public class Bulgarian implements Person {
  private String name;
  public Bulgarian(String name) {
    this.name = name;
  }
  public String getName() { return this.name; }
  public String sayHello() { return "Здравей"; }
}
// TODO: implement class Chinese
```



Abstract Classes

Abstract Class

- **Cannot** be instantiated
- May contain **abstract methods**
- Must provide an **implementation** for all **inherited** interface members
- Implementing an interface might map the interface methods onto **abstract** methods

```
public abstract class Animal {
}
```

Abstract Methods

Software University

- Declarations are only permitted in **abstract classes**
- Bodies must be **empty** (no curly braces)
- An abstract method declaration provides **no** actual implementation:

```
public abstract void build();
```

Interfaces vs Abstract Classes

Interface vs Abstract Class

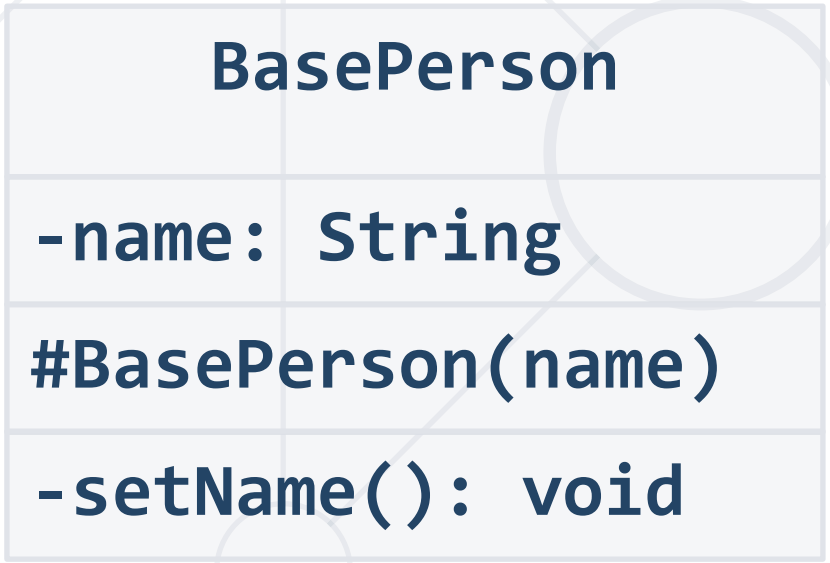
- Interface
  - A class may **implement several interfaces**
  - **Cannot have access modifiers**, everything is assumed as public
- Abstract Class (AC)
  - May **inherit only one abstract** class
  - **Provides implementation** and/or just the **signature** that has to be overridden
  - Can **contain access modifiers** for the fields, functions, properties

Interface vs Abstract Class

- Interface
  - If we add a **new method** we must **track down** **all** the **implementations** of the interface and **define implementation** for the new method
- Abstract Class
  - Fields and constants **can be defined**
  - If we add a **new method** we have the option of providing a **default implementation**

Problem: Say Hello Extended

- Refactor the code from the last problem
- Add **BasePerson abstract class**
  - In which move all **code duplication** from European, Bulgarian, Chinese



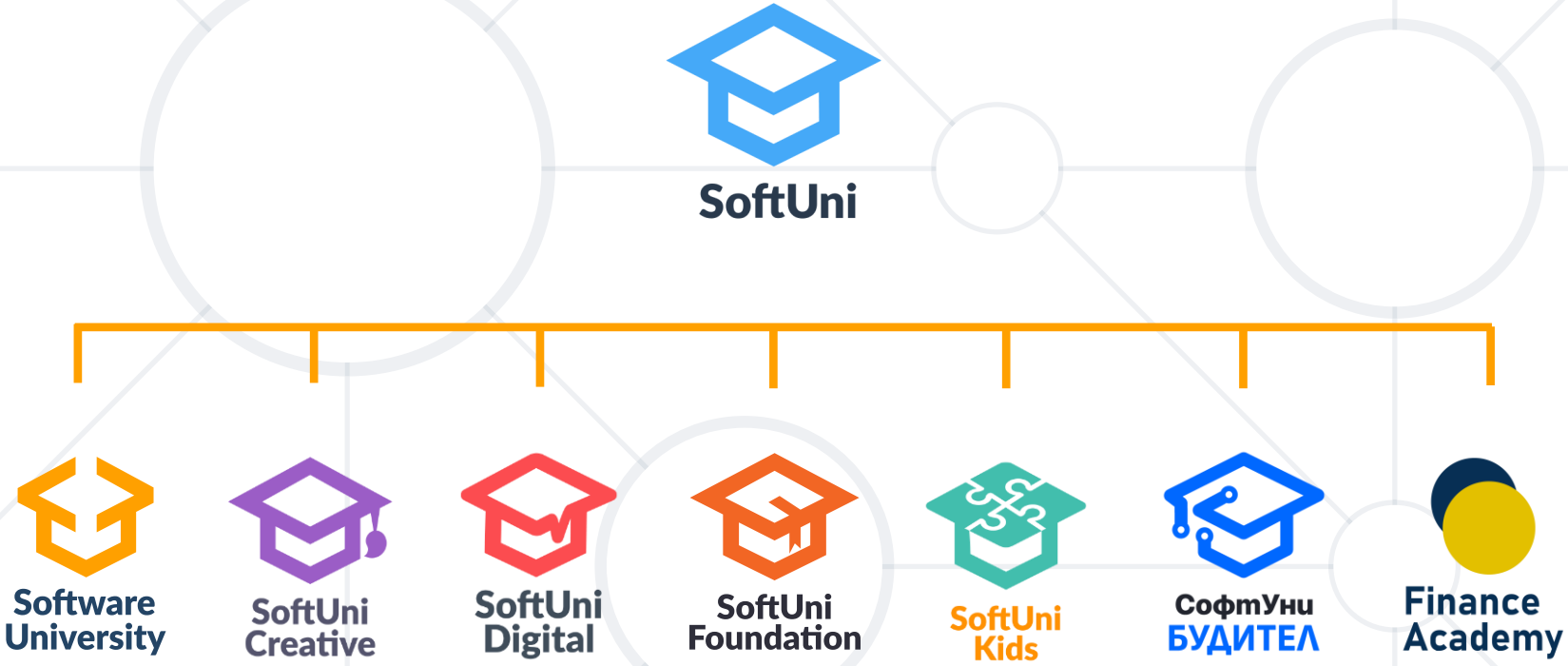
Solution: Say Hello Extended

```
public abstract class BasePerson implements Person {
  private String name;
  protected BasePerson(String name) {
    this.setName(name);
  }
  private void setName(String name) { this.name = name; }
  @Override
  public String getName() {
    return this.name;
  }
}
```

Summary

- Abstraction – hiding implementation and showing functionality
- Interfaces
  - implements vs extends
  - Default and Static methods
- Abstract classes
- Interfaces vs Abstract Classes

Questions?



SoftUni Diamond Partners

Software University

SmartIT

Coca-Cola HBC Bulgaria

SUPER HOSTING .BG

createX

Postbank

Решения за твоето утре

DRAFT KINGS

VIVACOM

AMBITIONED

INDEAVR

Serving the high achievers

SOFTWARE GROUP

PHAR VISION

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

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