Java OOP Principles

Abstraction, Interface, Inheritance, Polymorphism, Override / Overload

SoftUni Team

Technical Trainers

Software University

http://softuni.bg

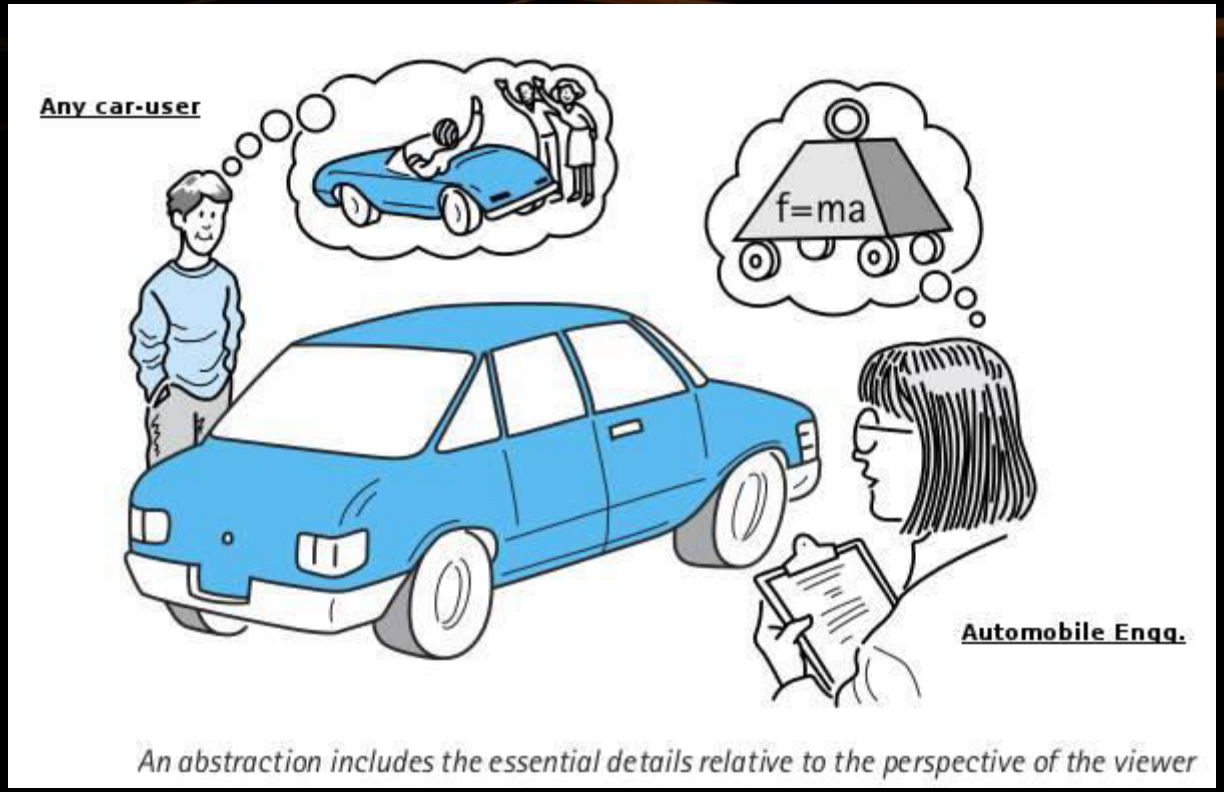
Table of Contents

Questions

SOFTWARE UNIVERSITY FOUNDATION

sli.do

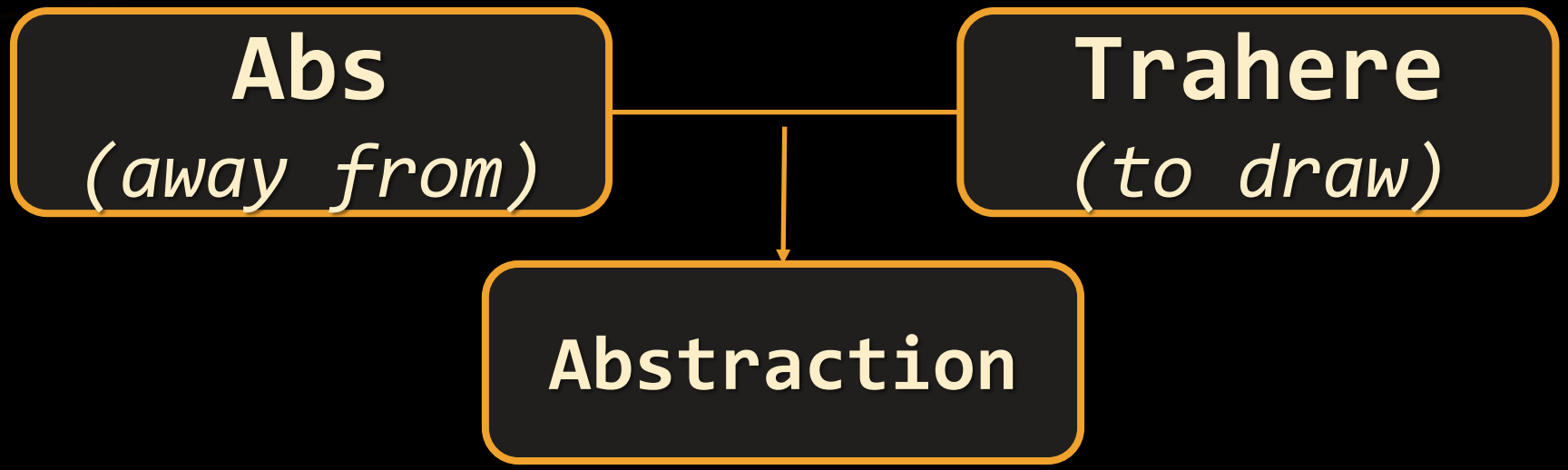
#JavaDB



Abstraction

What is Abstraction?

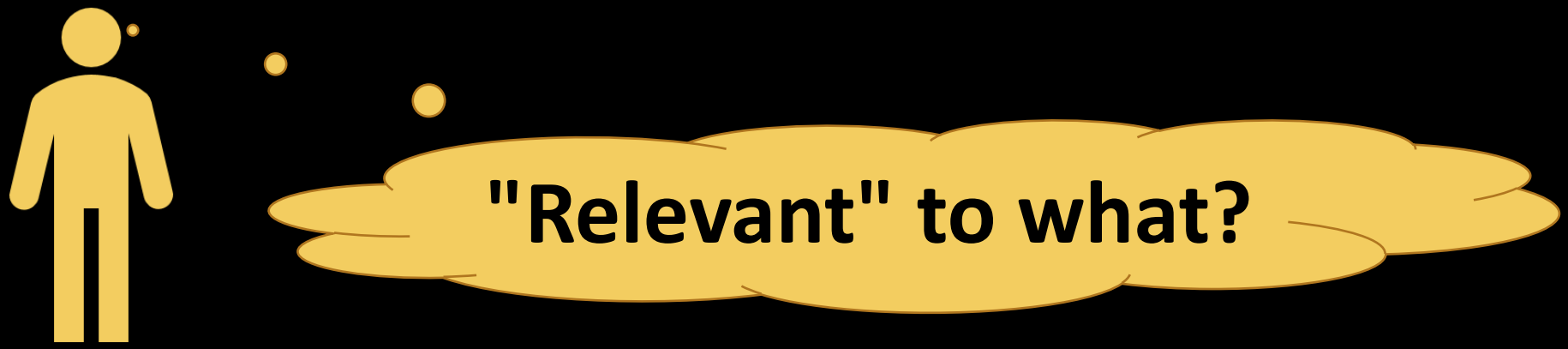
- From the Latin



Process of taking away or removing characteristics from something in order to reduce it to a set of essential characteristics.

Abstraction in OOP

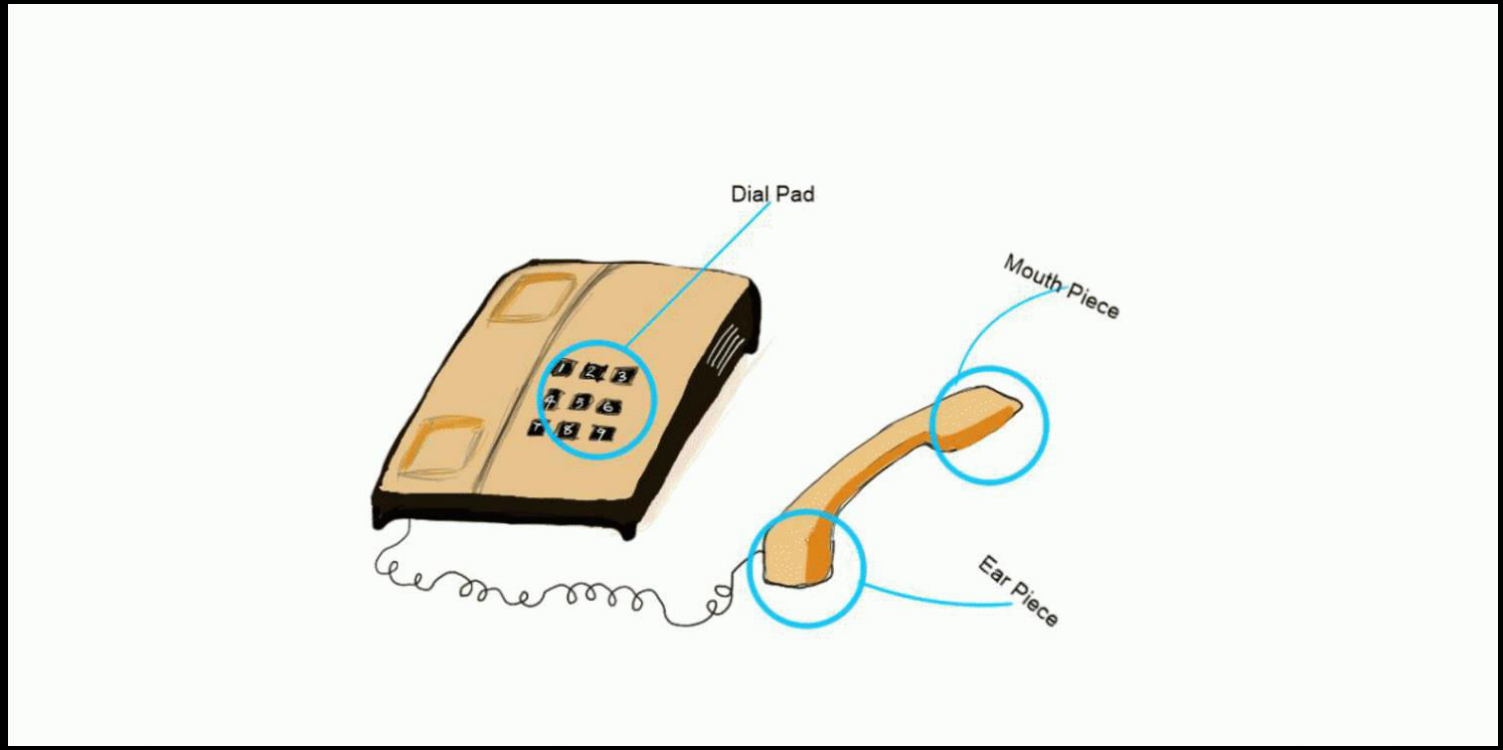
- Abstraction means ignoring irrelevant features, properties, or functions and emphasizing the relevant ones ...



- ... relevant to the project we develop
- Abstraction helps managing complexity

Abstraction Example

- Abstraction lets you focus on what the object does instead of how it does it.



How do we achieve abstraction?

- There are two 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

SOFTWARE UNIVERSITY FOUNDATION

Abstraction

- Achieve with interfaces and abstract classes
- Abstraction is a process of hiding the implementation details and showing only functionality to the user.

Encapsulation

- Achieve with access modifiers (private, public…)
- Encapsulation is used for hide the code and data in a single unit to protect the data from the outside the world

Abstraction vs Encapsulation (2)


PreSonus
AudioBox 22VSL
48V
Main
Phones
Power
1
2
Mic•Instrument
INST MIC
Clip
Inputs
VSL
Mixer
USB 2.0
MIDI
IN
OUT
MAIN OUT
R
L
PHONES

Interface

Interface

- Internal addition by compiler

Keyword

public or default modifier

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

Name

compiler

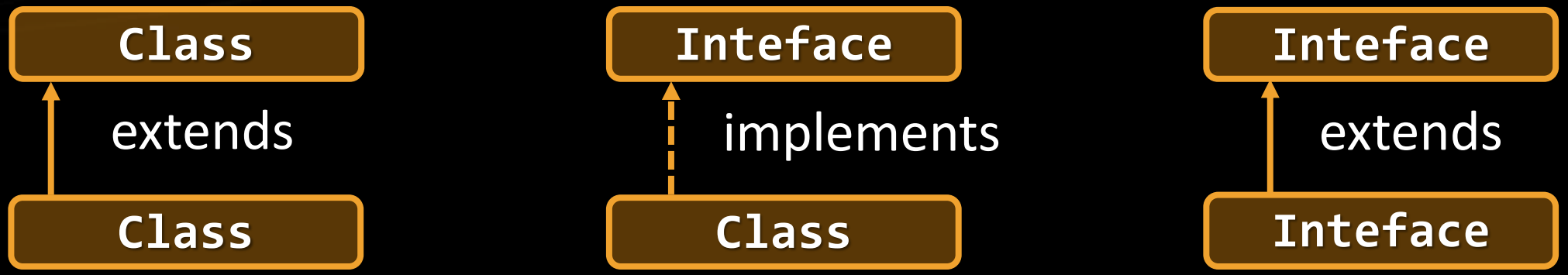
adds public static final before fields

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

public abstract before methods

implements vs extends

- Relationship between classes and interfaces



- Multiple inheritance

Interface Example

- Implementation of `print()` is provided in class A6

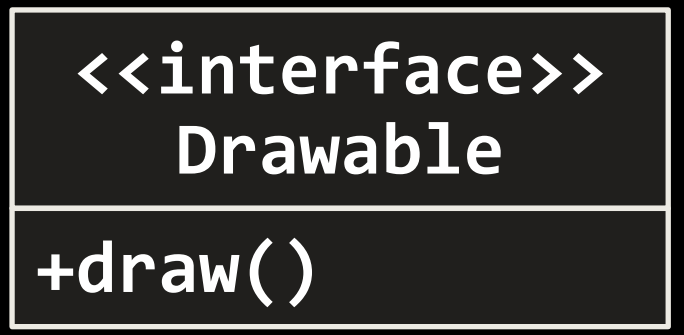
```
public interface Printable {
   int MIN = 5;
   void print();
}
```

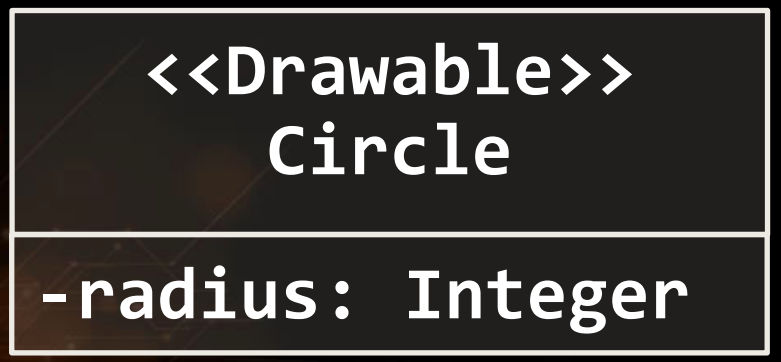
```
class Document implements Printable {
public void print() { System.out.println("Hello"); }

public static void main(String args[]){
Printable doc = new Document();
doc.print();
   }
}
```

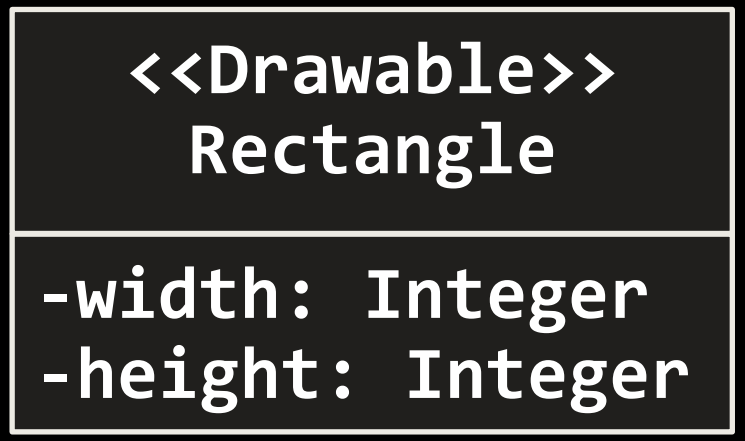
Polymorphism

Problem: Shapes Drawing

- Build project that contain interface for drawable objects
- Implements two type of shapes:
  Circle and Rectangle
- Both classes have to print on console
  their shape with "*".







Solution: Shapes Drawing

```
public interface Drawable {
  void draw();
}
```

```
public class Rectangle implements Drawable {
  //TODO Add fields and constructor
  @Override
  public void draw() { slide 17 } }
```

```
public class Circle implements Drawable {
  //TODO Add fields and constructor
  @Override
  public void draw() { slide 18 } }
```

Solution: Shapes Drawing - Rectangle Draw


```
Public class Rectangle implements Drawable {
public void draw() {
  for (int i = 0; i < height; i++) {
    System.out.print("*");
    for (int k = 1; k < width - 1; k++) {
      System.out.print(" ");
      if (i == 0 || i == (height - 1)) {
        System.out.print("*");
      } else {
        System.out.print(" ");
      } }
    System.out.print(" "); System.out.print("*");
    System.out.print("\n"); } } }
```

Solution: Shapes Drawing - Circle Draw


```
public class Circle implements Drawable {
  public void draw() {
    double r_in = this.radius - 0.4;
    double r_out = this.radius + 0.4;
    for(double y = this.radius; y >= -this.radius; --y) {
      for(double x = -this.radius; x < r_out; x += 0.5) {
        double value = x * x + y * y;
        if(value >= r_in * r_in && value <= r_out * r_out) {
          System.out.print("*");
        } else {
          System.out.print(" ");
        } }
    System.out.println(); } } }
```

Problem: Car Shop

Solution: Car Shop

```
public interface Car {
    int TIRES = 4;

    String getModel();

    String getColor();

    int getHorsePower();
}
```

Check your solution here: https://judge.softuni.bg/Contests/Compete/Index/498#3

Solution: Car Shop

```
public class Seat implements Car, Serializable {
  //TODO: Add fields
  //TODO: Add constructor
  //TODO: Add private methods

  @Override
  public String getModel() { return this.model; }
  @Override
  public String getColor() { return this.color; }
  @Override
  public int getHorsePower() { return this.horsePower; }
}
```

Check your solution here: https://judge.softuni.bg/Contests/Compete/Index/498#3

SOFTWARE UNIVERSITY FOUNDATION

Interfaces

Live Exercises in Class (Lab)

Extend Interface

- Interface can extend another interface

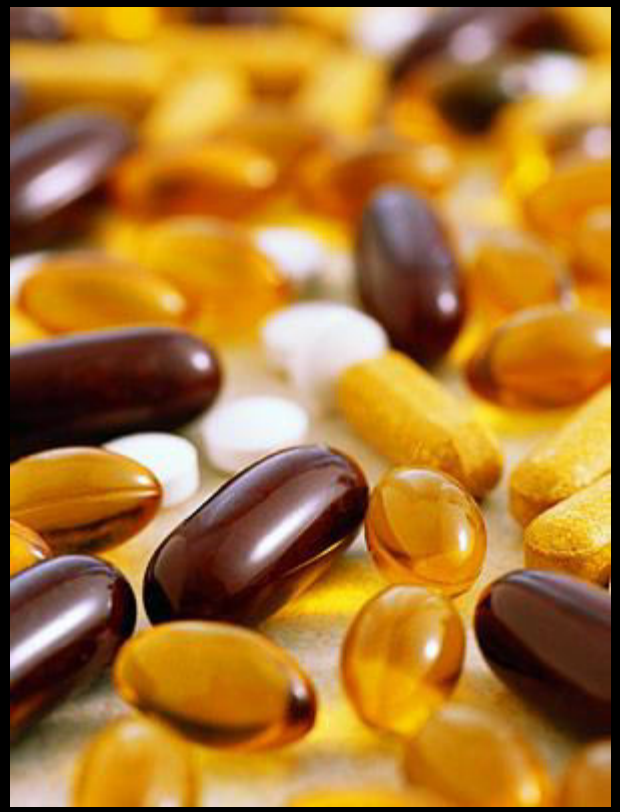
```
public interface Showable  {
   int MIN = 5;
   void show();
}
```

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

Extend Interface (2)

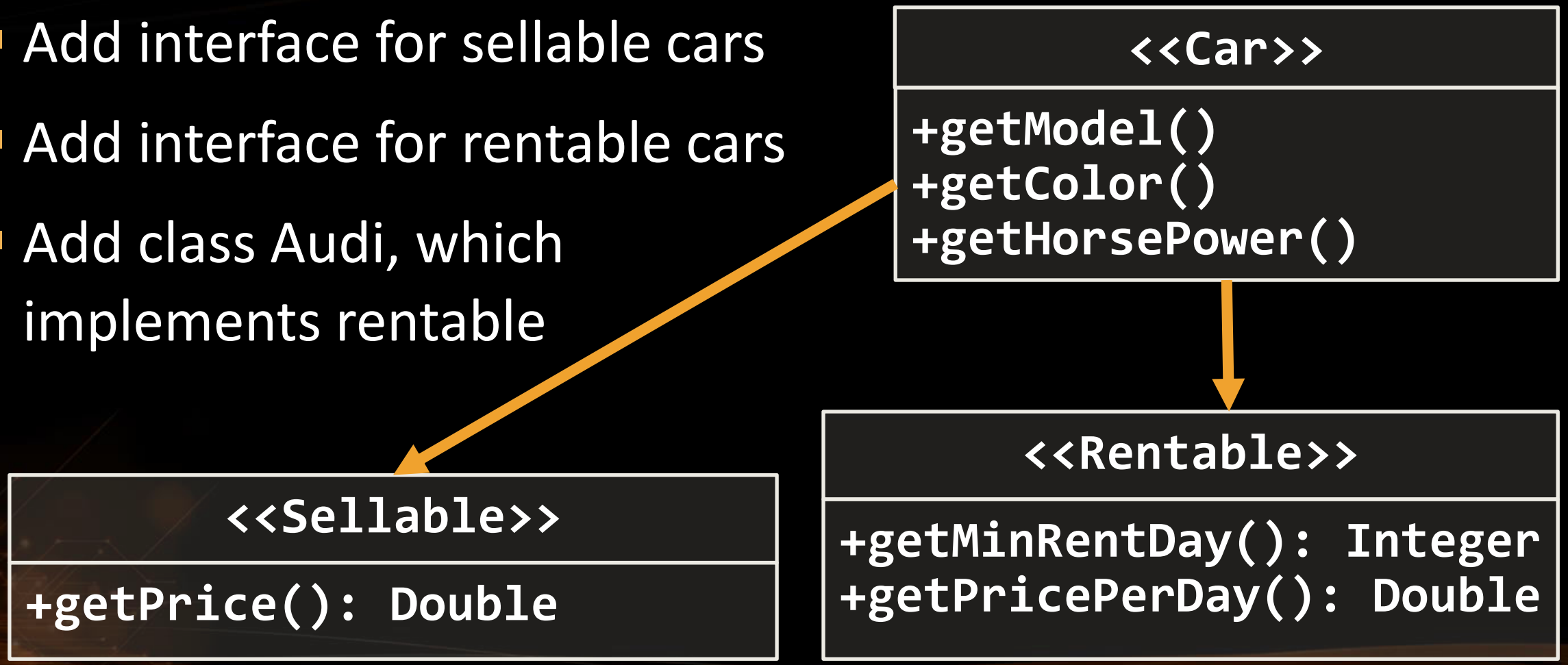
- Class which implements child interface MUST provide implementation for parent interface too

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

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



Problem: Car Shop

- Refactor your first problem code
- Add interface for sellable cars
- Add interface for rentable cars
- Add class Audi, which implements rentable



Solution: Car Shop

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

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

    Double getPricePerDay();
}
```

Check your solution here: https://judge.softuni.bg/Contests/Compete/Index/498#3

Solution: Car Shop

```
public class Audi implements Rentable {
  @Override
  public String getModel() { return this.model; }
  @Override
  public String getColor() { return this.color; }
  @Override
  public int getHorsePower() { return this.horsePower; }
  @Override
  public Integer getMinRentDay() {
    return this.minDaysForRent; }
  @Override
  public Double getPricePerDay() {
    return this.pricePerDay; } }
```

Default Method

- Since Java 8, we can have method body in interface

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

- If you need to Override default method think about you Design

Default Method (2)

- Implementation doesn't need for default methods

```
class Rectangle implements Drawable{
public void draw() {
  System.out.println("drawing rectangle"); }
}
```

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

```
//drawing rectangle
//default method
```

Static Method

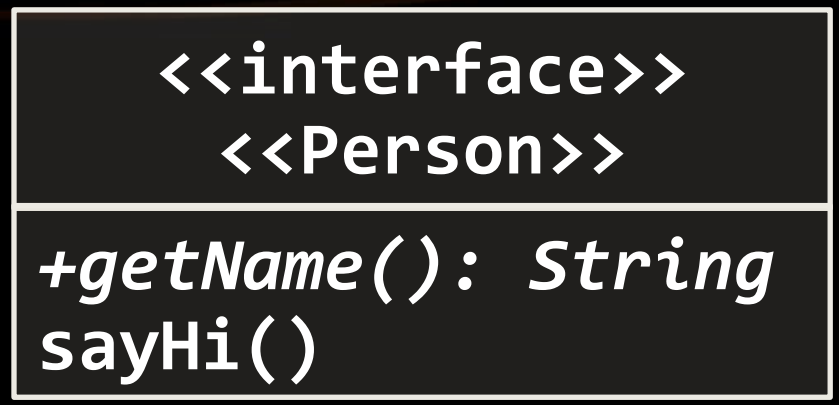
- Since Java 8, we can have static method in interface

```
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 Hi!

- Design project, which have:
  - Interface for Person
  - Three implementation for different nationality
  - Override where need



Solution: Say Hello

```
public interface Person {
   String getName();

   default void sayHello() { System.out.println("Hello");}
}
```

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

Check your solution here: https://judge.softuni.bg/Contests/Compete/Index/498#3

Solution: Say Hello

```
public class Bulgarian implements Person {
   private String name;
   public Bulgarian(String name) {
      this.name = name;
   }
   @Override
   public String getName() { return this.name; }
   @Override
   public void sayHello() {System.out.println("Здравей");}
}
//TODO: Make same for Chinese
```

Check your solution here: https://judge.softuni.bg/Contests/Compete/Index/498#3

Interface vs Abstract Class

SOFTWARE UNIVERSITY FOUNDATION

Abstract Class

- Abstract class **doesn't support multiple inheritance**.
- Abstract class can **have abstract and non-abstract** methods.
- Abstract class **can have final, non-final, static and non-static variables**.

Interface

- Interface supports multiple inheritance.
- Interface can have **only abstract** methods.
  Since Java 8, it can have **default and static methods** also.
- Interface has **only static and final variables**.

Problem: Base Person

- Refactor code from last problem
- Add BasePerson abstract class
  - Move in it all code duplication from European, Bulgarian, Chinese

Solution: Say Hello

```
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;
   } }
```

Check your solution here: https://judge.softuni.bg/Contests/Compete/Index/498#3

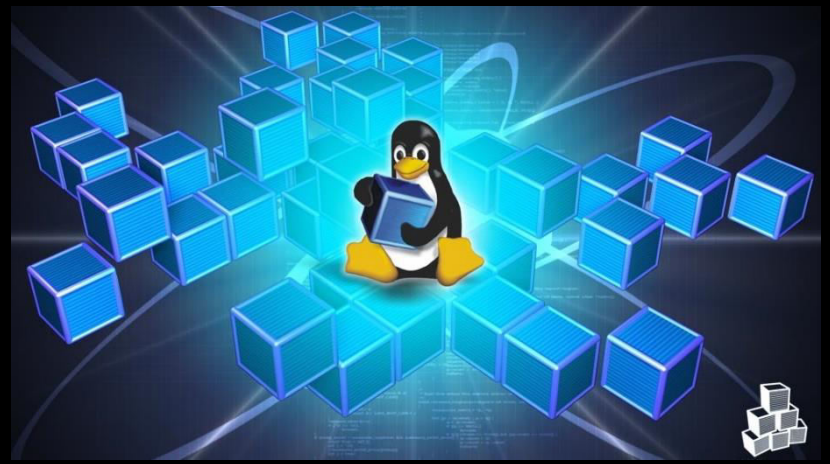
SOFTWARE UNIVERSITY FOUNDATION

Interfaces and Abstract Class

Live Exercises in Class (Lab)

Summary

1. Abstraction
2. Interface
3. Inheritance
4. Polymorphism
5. Override / Overload



Java OOP Principles

Questions?

https://softuni.bg/courses/databases-advanced-hibernate

License

- This course (slides, examples, demos, videos, homework, etc.) is licensed under the "Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International" license

- Attribution: this work may contain portions from
  - "Databases" course by Telerik Academy under CC-BY-NC-SA license

Free Trainings @ Software University

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