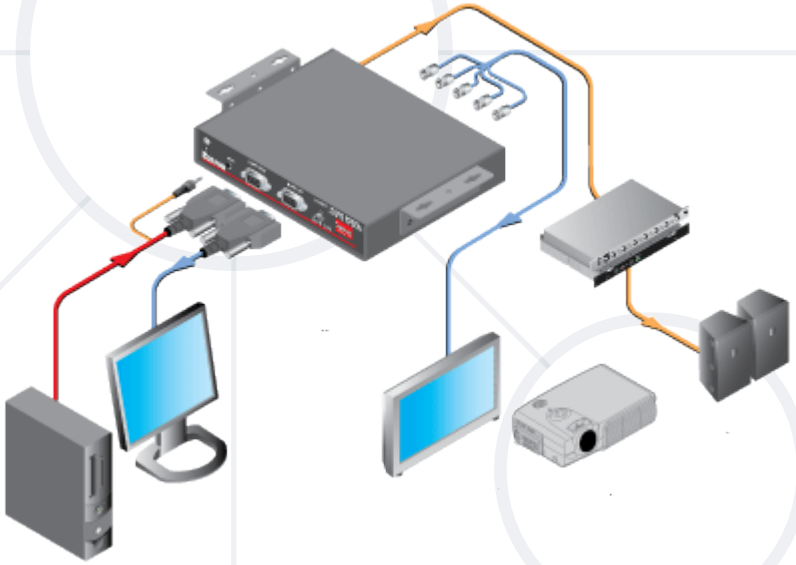
Interfaces and Abstraction

Interfaces vs Abstract Classes
Abstraction vs Encapsulation



SoftUni Team

http://softuni.bg

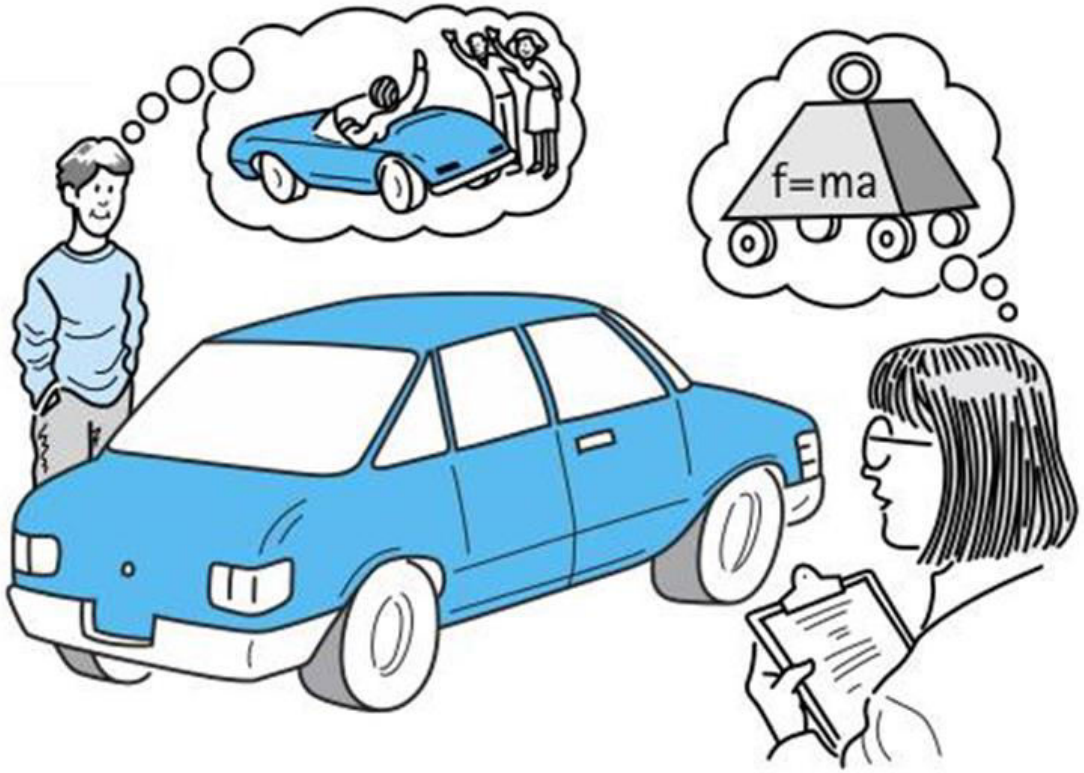
Table of Contents

Have a Question?

SoftUni Foundation

sli.do

#java-advanced

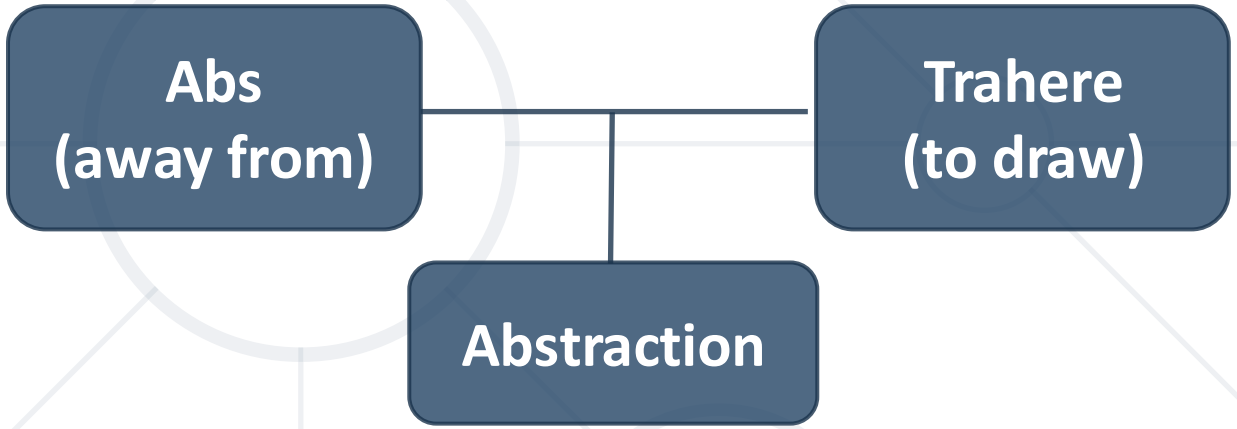


Abstraction

Working with Abstraction

What is Abstraction?

- From the Latin



- **Preserving information** that is **relevant** in a given context, and **forgetting information** that is **irrelevant** in that context

Abstraction in OOP

SoftUni Foundation

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

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

How Do We Achieve Abstraction?

SoftUni Foundation

- 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

- 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

Working with Interfaces

Interface

- Internal addition by compiler

Keyword

Public or default modifier

Name

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

compiler

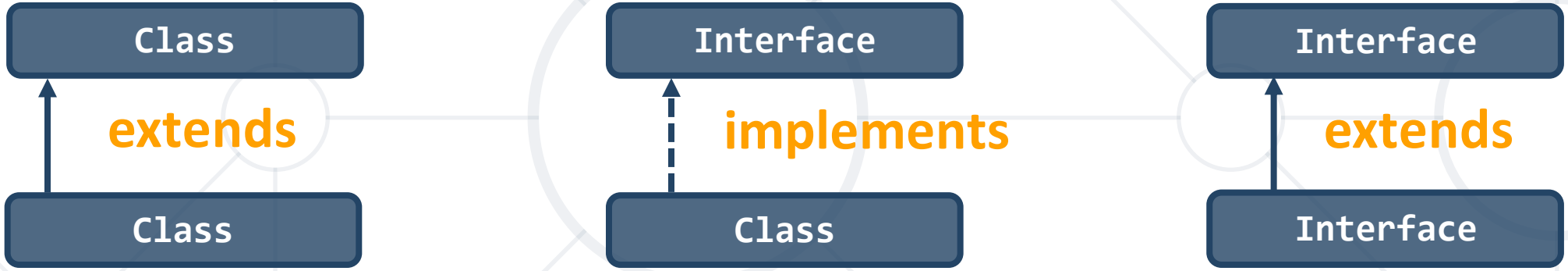
Adds public static final before fields

Public abstract before methods

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

implements vs extends

- Relationship between classes and interfaces



- Multiple inheritance

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://judge.softuni.bg/Contests/1581/Interfaces-and-Abstraction-Lab

Solution: Car Shop

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

Check your solution here :https://judge.softuni.bg/Contests/1581/Interfaces-and-Abstraction-Lab

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; }
}
```

Check your solution here :https://judge.softuni.bg/Contests/1581/Interfaces-and-Abstraction-Lab

Extend Interface

- Interface can **extend another interface**

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

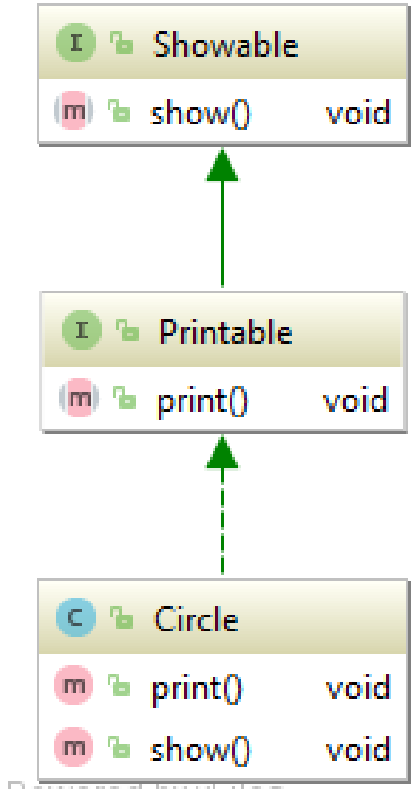
```
public interface Printable extends Showable {
  void print();
}
```

Extend Interface

- 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 Extended

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

Check your solution here :https://judge.softuni.bg/Contests/1581/Interfaces-and-Abstraction-Lab

Solution: Car Shop Extended (1)

```
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/1581/Interfaces-and-Abstraction-Lab

Solution: Car Shop Extended (2)

```
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
}
```

Check your solution here :https://judge.softuni.bg/Contests/1581/Interfaces-and-Abstraction-Lab

Default Method

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

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

- If you need to Override 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 **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 Hello

- Design a project, which has:
  - **Interface** for Person
  - Three implementation for different nationalities
  - Override where needed

```
<<interface>>
<<Person>>
+getName(): String
+sayHello():String
```

```
<<Person>>
Bulgarian
-name: String
+sayHello(): String
```

```
<<Person>>
European
-name: String
```

```
<<Person>>
Chinese
-name: String
+sayHello(): String
```

Check your solution here :https://judge.softuni.bg/Contests/1581/Interfaces-and-Abstraction-Lab

Solution: Say Hello

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

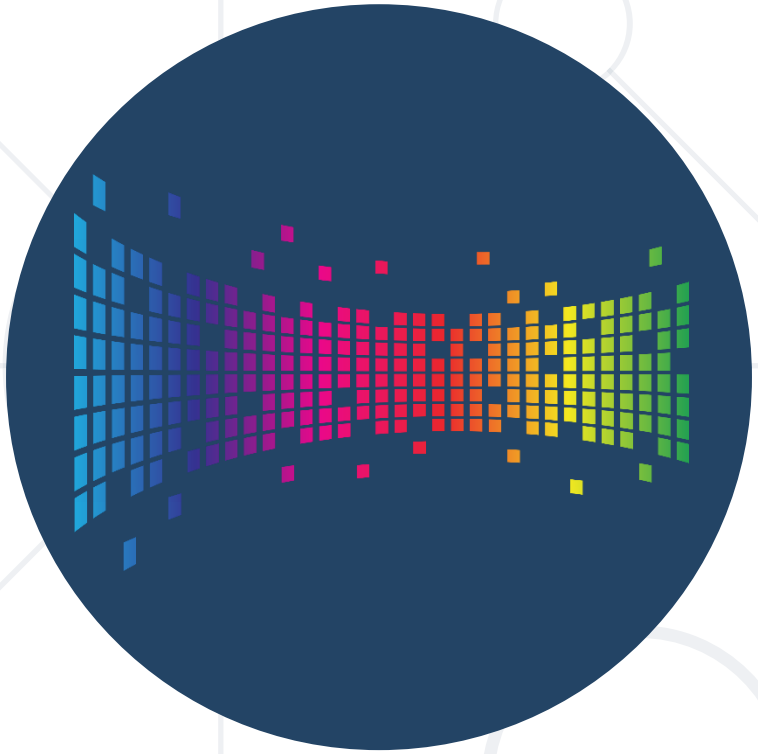
```
public class European implements Person {
  private String name;
  public European(String name) { this.name = name; }
  public String getName() { return this.name; }
}
```

Check your solution here :https://judge.softuni.bg/Contests/1581/Interfaces-and-Abstraction-Lab

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
```

Check your solution here :https://judge.softuni.bg/Contests/1581/Interfaces-and-Abstraction-Lab



Abstract Classes

Abstract Classes and Methods

Abstract Class

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

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

Abstract Methods

- 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 (1)

- 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
  - Can **provide implementation** and/or just the **signature** that have to be overridden
  - **Can contain access modifiers** for the fields, functions, properties

Interface vs Abstract Class (2)

SoftUni Foundation

- Interface
  - If we add **a new method we have to 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 default implementation** and therefore all the existing code might work properly

Software University

Problem: Say Hello Extended

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

BasePerson
-name: String
#BasePerson(name)
-setName(): void

Check your solution here :https://judge.softuni.bg/Contests/1581/Interfaces-and-Abstraction-Lab

Solution: Say Hello Extended

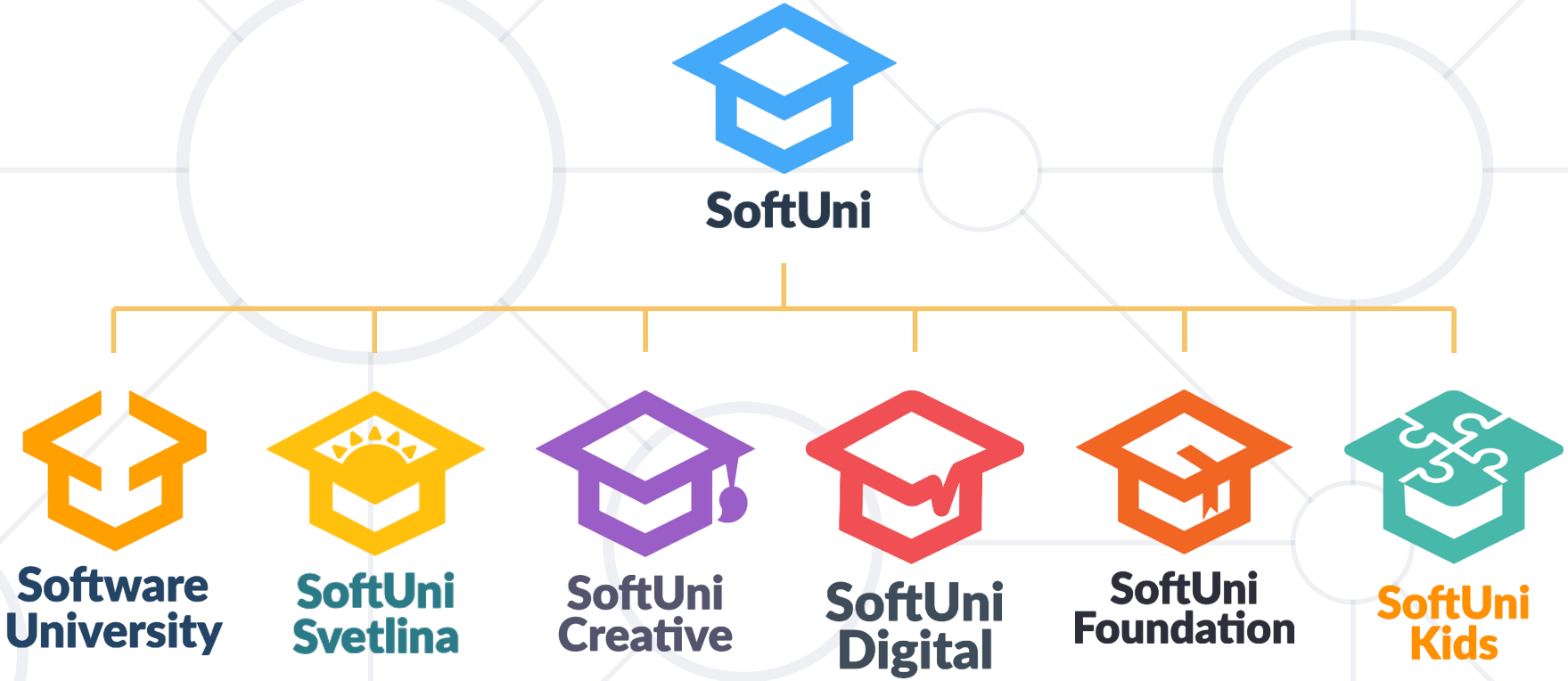
```
public abstract class BasePerson implements Person {
  private String name;
  protected BasePerson(String name) {
    this.setName(name);
  }
  private void setName(String name) { this.name = name; }
  @Override
  public String getName() {
    return this.name;
  }
}
```

Check your solution here :https://judge.softuni.bg/Contests/1581/Interfaces-and-Abstraction-Lab

Summary

- Abstraction
- Interfaces
  - Implements vs Extends
  - Default and Static methods
- Abstract classes
- Interfaces vs Abstract Classes

Questions?


SoftUni
Software University
SoftUni Svetlina
SoftUni Creative
SoftUni Digital
SoftUni Foundation
SoftUni Kids

SoftUni Diamond Partners

SoftUni Foundation

XSsoftware

SBTech
we know sports

telenor

SoftwareGroup
doing it right

NETPEAK

SmartIT

Postbank
Решения за твоето утре

SUPER HOSTING .BG

INDEAVR
Serving the high achievers

INFRAGISTICS®

STEMO®
Computer Systems & Software

SoftUni Organizational Partners

SoftUni Foundation

ИНФОРМАЦИОННО ОБСЛУЖВАНЕ

OneBit SOFTWARE

Lukanet.com

WORLD OF MYTHS

Trainings @ Software University (SoftUni)

- Software University – High-Quality Education and Employment Opportunities
  - softuni.bg
- Software University Foundation
  - http://softuni.foundation/
- Software University @ Facebook
  - facebook.com/SoftwareUniversity
- Software University Forums
  - forum.softuni.bg

License

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