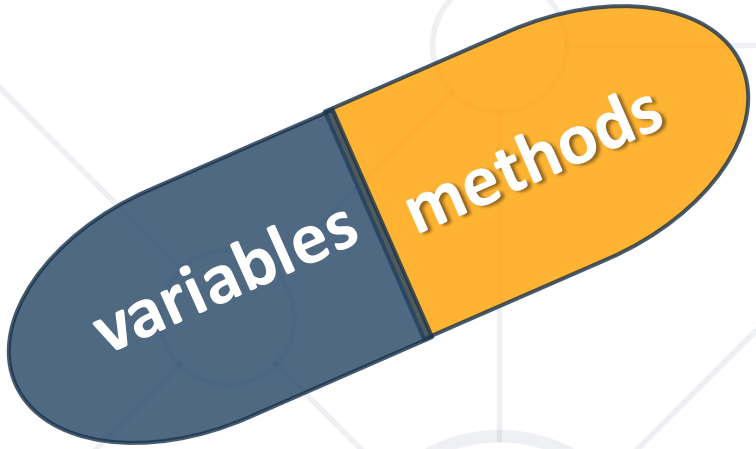
Encapsulation

Benefits of Encapsulation



SoftUni Team

Technical Trainers

Software University

SoftUni

Software University

https://softuni.bg

Have a Question?

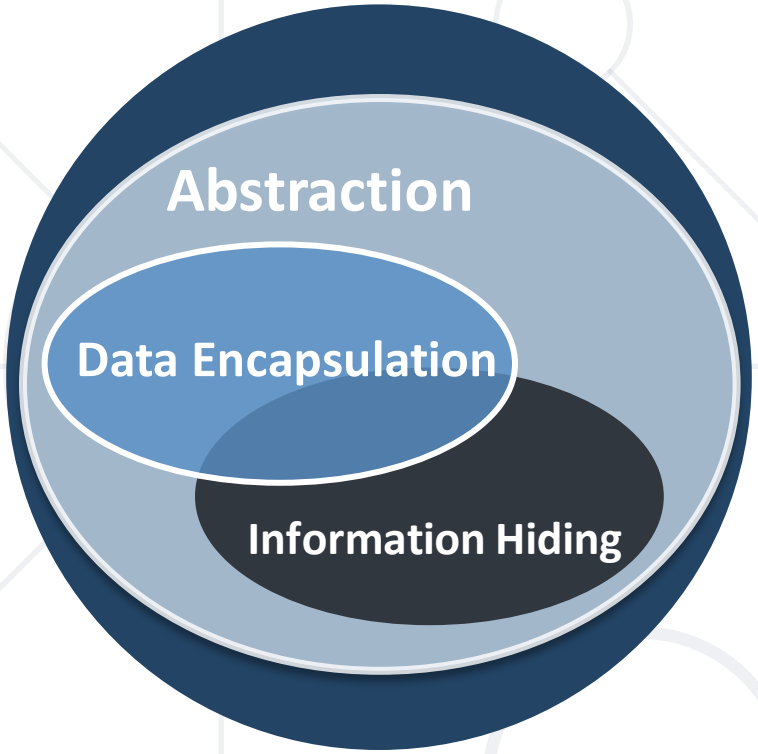
Software University

sli.do

#java-advanced

Table of Contents

1. What is **Encapsulation**?
   - Keyword **this**
2. **Access** Modifiers
3. **Validation**
4. **Mutable** and **Immutable** Objects
5. Keyword **final**



Hiding Implementation

Encapsulation

- Process of wrapping code and data together into a single unit
- Flexibility and extensibility of the code
- Reduces **complexity**
- Structural changes remain **local**
- Allows **validation** and **data binding**

Encapsulation

- Objects fields **must be private**

```
class Person {
   private int age;
}
```

- Use **getters** and **setters** for data access

```
class Person {
   public int getAge()
   public void setAge(int age)
}
```

Encapsulation – Example

- Fields should be **private**

```
                Person
-name: string
-age: int
+Person(String name, int age)
+getName(): String
+setName(String name): void
+getAge(): int
+setAge(int age): void
```

- == private

+ == public

- Accessors and Mutators should be **public**

Keyword This

- **this** is a reference to the **current object**
- **this** can refer to current class instance

```
public Person(String name) {
   this.name = name;
}
```

- **this** can invoke the current class method

```
public String fullName() {
   return this.getFirstName() + " " + this.getLastName();
}
```

Keyword This

- **this** can invoke a current class constructor

```
public Person(String name) {
   this.firstName = name;
}
```

```
public Person (String name, Integer age) {
   this(name);
   this.age = age;
}
```

Access Modifiers

Private Access Modifier

- Object hides data from the outside world

```
class Person {
   private String name;
   Person (String name) {
      this.name = name;
   }
}
```

- Classes and interfaces **cannot** be private
- Data can be **accessed only within the declared class** itself

Protected Access Modifier

- Grants **access to subclasses**

```
class Team {
   protected String getName () {…}
   protected void setName (String name) {…}
}
```

- The **protected** modifier cannot be applied to classes and interfaces
- Prevents a **nonrelated** class from trying to use it

Default Access Modifier

- Do not explicitly declare an access modifier

```
class Team {
   String getName() {…}
   void setName(String name) {…}
}
```

- **Available** to any other class in the same **package**

```
Team real = new Team("Real");
real.setName("Real Madrid");
System.out.println(real.getName());
                          // Real Madrid
```

Public Access Modifier

- Grants access to **any class** belonging to the **Java Universe**

```
public class Team {
  public String getName() {…}
  public void setName(String name) {…}
}
```

- Import a package if you need to use a class
- The `main()` method of an application must be **public**

Problem: Sort by Name and Age

- Create a class **Person**

Person
-firstName: String -lastName: String -age: int
+getFirstName(): String +getLastName(): String +getAge(): int +toString(): String

```
Collections.sort(persons, (firstPerson, secondPerson) -> {
    int sComp = firstPerson
                .getFirstName()
                .compareTo(secondPerson.getFirstName());

    if (sComp != 0) {
        return sComp;
    } else {
        return firstPerson
                .getAge()
                .compareTo(secondPerson.getAge());
    }
});
```

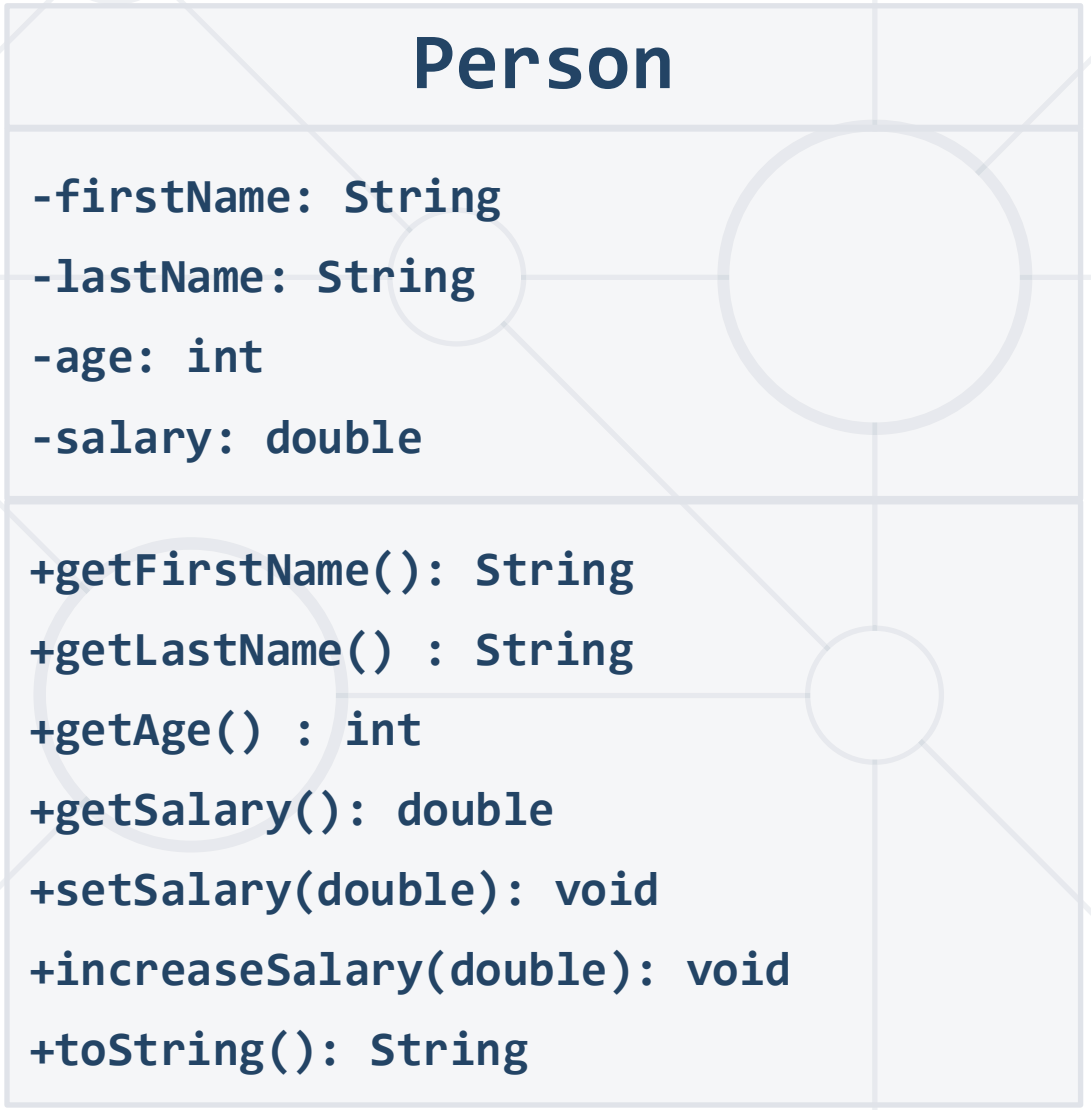
Check your solution here: https://judge.softuni.bg/Contests/1535/Encapsulation-Lab

Solution: Sort by Name and Age

```
public class Person {
  private String firstName;
  private String lastName; private int age;
  // TODO: Implement Constructor

  public String getFirstName() { /* TODO */ }

  public String getLastName() { /* TODO */ }

  public int getAge() { return age; }

  @Override
  public String toString() { /* TODO */ }
}
```

Problem: Salary Increase

- Implement Salary
- Add:
  - getter for salary
  - increaseSalary by percentage
- Persons younger than 30 get **only half** of the increase



Solution: Salary Increase

- Expand Person from previous task

```
public class Person {
  private double salary;
  // Edit Constructor
  public double getSalary() {
    return this.salary;
  }
  public void setSalary(double salary) {
    this.salary = salary;
  }
  // Next Slide…
  // TODO: Edit toString() method
}
```

Check your solution here: https://judge.softuni.bg/Contests/1535/Encapsulation-Lab

Solution: Salary Increase

- Expand Person from previous task

```
public void increaseSalary(double percentage) {
  if (this.getAge() < 30) {
    this.setSalary(this.getSalary() +
                   (this.getSalary() * percentage / 200));
  } else {
    this.setSalary(this.getSalary() +
                   (this.getSalary() * percentage / 100));
  }
}
```

Check your solution here: https://judge.softuni.bg/Contests/1535/Encapsulation-Lab

Validation

Validation

- **Data validation** happens in **setters**

```
private void setSalary(double salary) {
  if (salary < 460) {
    throw new IllegalArgumentException("Message");
  }

  this.salary = salary;
}
```

It is better to throw **exceptions**, rather than printing to the Console

- Printing with **System.out couples** your class
- The **Client** can **handle** class exceptions

Validation

- Constructors use **private setters** with validation logic

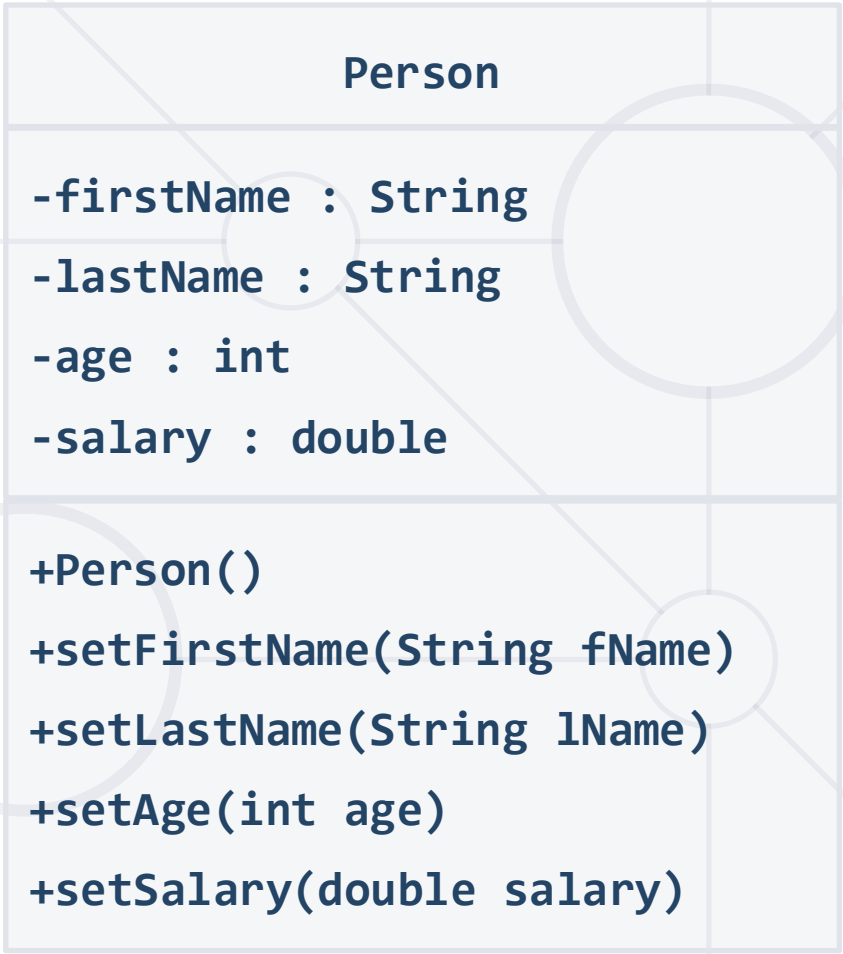
```
public Person(String firstName, String lastName,
              int age, double salary) {
  setFirstName(firstName);
  setLastName(lastName);
  setAge(age);
  setSalary(salary);
}
```

Validation happens inside the setter

- Guarantees **valid state** of an object in its creation
- Guarantees **valid state** for public setters

Problem: Validation Data

- Expand Person with **validation for every field**
- Names should be **at least 3 symbols**
- Age **cannot be zero or negative**
- Salary **cannot be less than 460**



Check your solution here: https://judge.softuni.bg/Contests/1535/Encapsulation-Lab

Solution: Validation Data

```
// TODO: Add validation for firstName
// TODO: Add validation for lastName
public void setAge(int age) {
   if (age < 1) {
      throw new IllegalArgumentException(
         "Age cannot be zero or negative integer");
   }
   this.age = age;
}
// TODO: Add validation for salary
```

Check your solution here: https://judge.softuni.bg/Contests/1535/Encapsulation-Lab

Mutable and Immutable Objects

Mutable vs Immutable Objects

- Mutable Objects
  - The contents of that instance **can** be altered

```
Point myPoint = new Point(0, 0);
myPoint.setLocation(1.0, 0.0);
System.out.println(myPoint);
```

```
java.awt.Point[1.0, 0.0]
```

- Immutable Objects
  - The contents of the instance **can't** be altered

```
String str = new String("old String");
System.out.println(str);
str.replaceAll("old", "new");
System.out.println(str);
```

```
old String
old String
```

Mutable Fields

- **private** mutable fields are not fully encapsulated

```
class Team {
   private String name;
   private List<Person> players;

   public List<Person> getPlayers() {
      return this.players;
   }
}
```

- In this case, the **getter is like a setter too**

Mutable Fields – Example

```
Team team = new Team();
Person person = new Person("David", "Adams", 22);
team.getPlayers().add(person);
System.out.println(team.getPlayers().size());  // 1
team.getPlayers().clear();
System.out.println(team.getPlayers().size());  // 0
```

Imutable Fields

- For securing our collection we can return **Collections.unmodifiableList()**

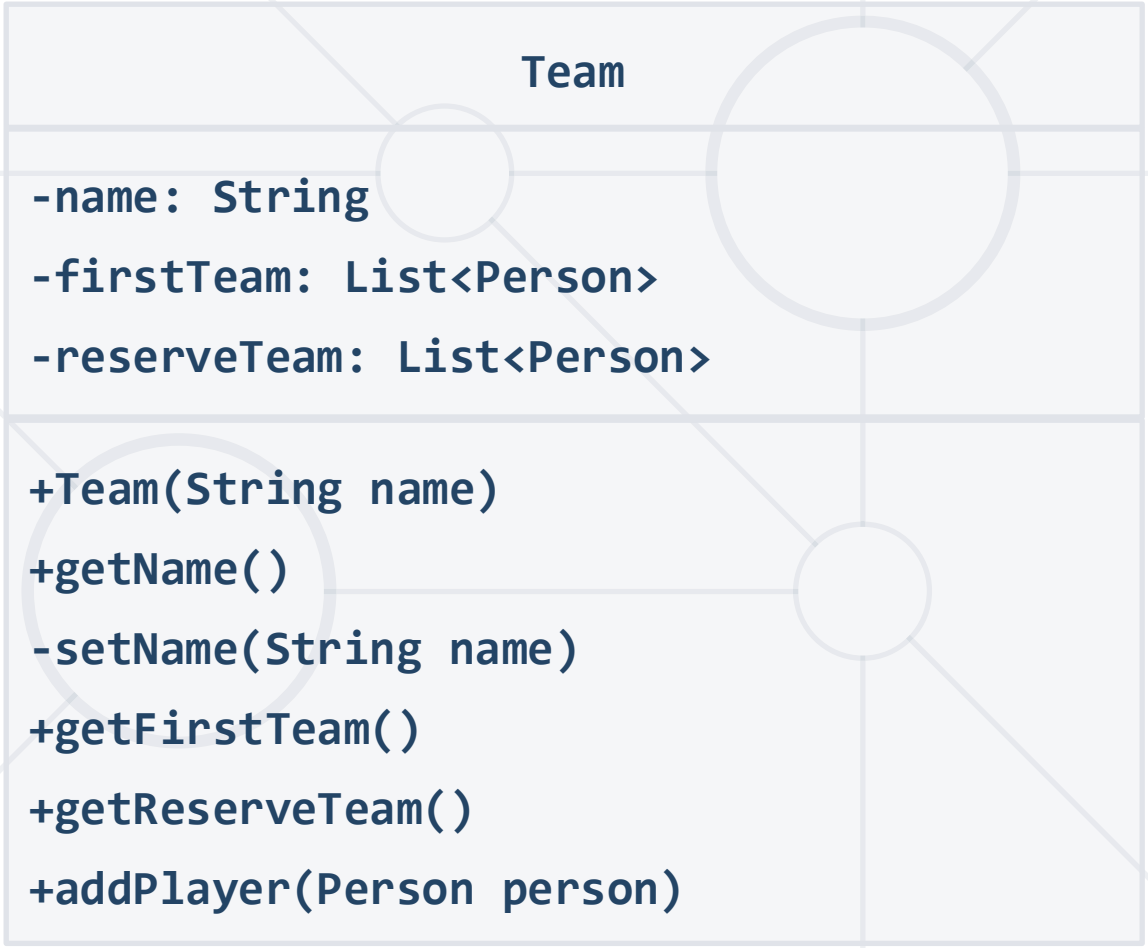
```
class Team {
  private List<Person> players;

  public void addPlayer(Person person) {
    this.players.add(person);
  }
  public List<Person> getPlayers() {
    return Collections.unmodifiableList(players);
  }
}
```

Add new methods for functionality over list

Returns a safe collections

Problem: First and Reserve Team

- Expand your project with class **Team**
- Team have two squads **first team** and **reserve team**
- Read persons from console and **add** them to team
- If they are **younger** than **40**, they go to **first squad**
- **Print** both squad **sizes**



Check your solution here: https://judge.softuni.bg/Contests/1535/Encapsulation-Lab

Solution: First and Reserve Team

```
private List<Person> firstTeam;
private List<Person> reserveTeam;

public void addPlayer(Person person) {
  if (person.getAge() < 40)
    this.firstTeam.add(person);
  else
    this.reserveTeam.add(person);
}
public List<Person> getFirstTeam() {
  return Collections.unmodifiableList(firstTeam);
}
// TODO: add getter for reserve team
```

Check your solution here: https://judge.softuni.bg/Contests/1535/Encapsulation-Lab

Keyword Final

Keyword Final

- A **final class** can't be extended

```
public class Animal {}
public final class Mammal extends Animal {}
public class Cat extends Mammal {}
```

- A **final method** can't be overridden

```
public final void move(Point point) {}
public class Mammal extends Animal {
   @Override
   public void move() {}
}
```

Keyword Final

- The `final variable` value can't be changed once it is set

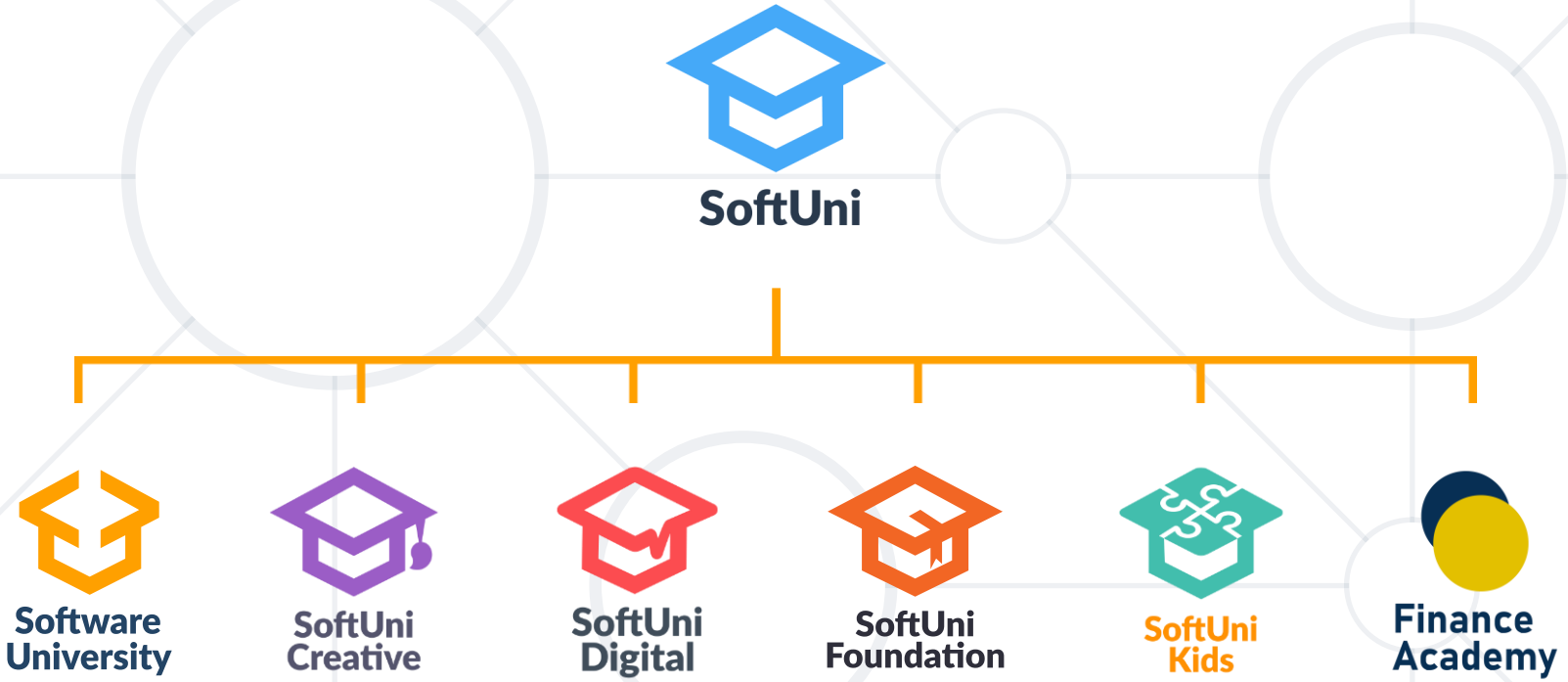
```
private final String name;
private final List<Person> firstTeam;

public Team (String name) {
  this.name = name;
  this.firstTeam = new ArrayList<Person> ();
}
public void doSomething(Person person) {
  this.name = "";
  this.firstTeam = new ArrayList<>();
  this.firstTeam.add(person);
}
```

Compile time error

Summary

- Encapsulation:
  - Hides implementation
  - Reduces complexity
  - Ensures that structural changes remain local
- Mutable and Immutable objects
- Keyword final

Questions?



SoftUni Diamond Partners

Software University

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