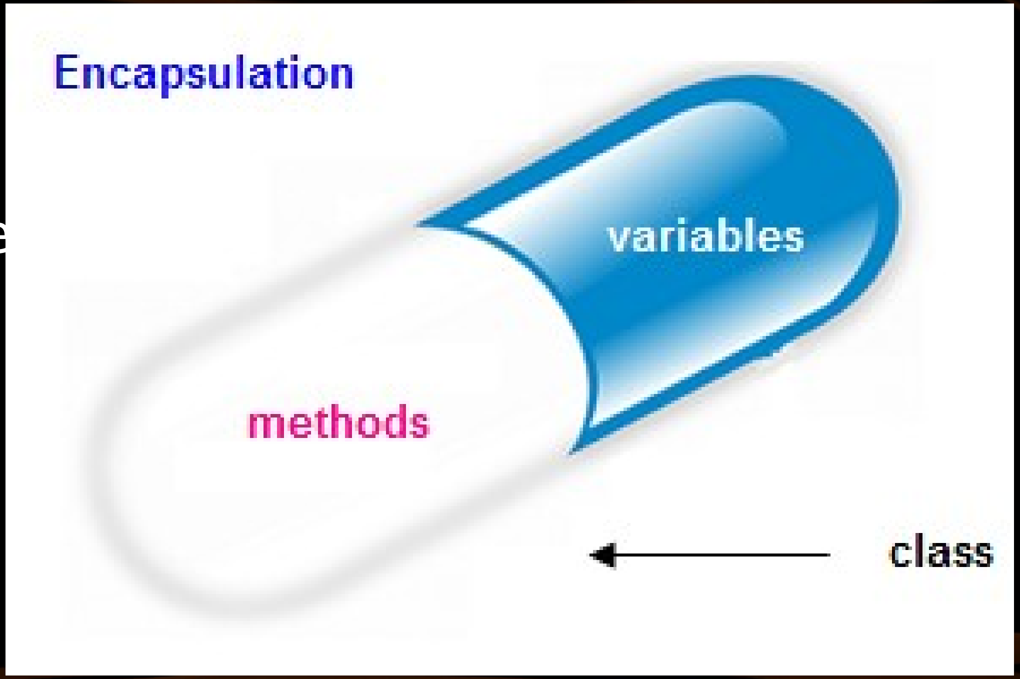
Encapsulation

What is Encapsulation, Benefits, Implementation in Java

Presentation Subtitle

Java OOP Basics

a Picture He



- SoftUni Team
- Technical Trainers
- Software University
- http://softuni.bg

Table of Contents

1. What is Encapsulation?
2. Keyword `this`
3. Access Modifiers
4. Validation
5. Mutable and Immutable objects
6. Keyword `final`

Questions

sli.do

#Java-OOP

Encapsulation

- Hiding Implementation

Encapsulation

- Process of wrapping code and data together into a single unit
- Objects fields must be private

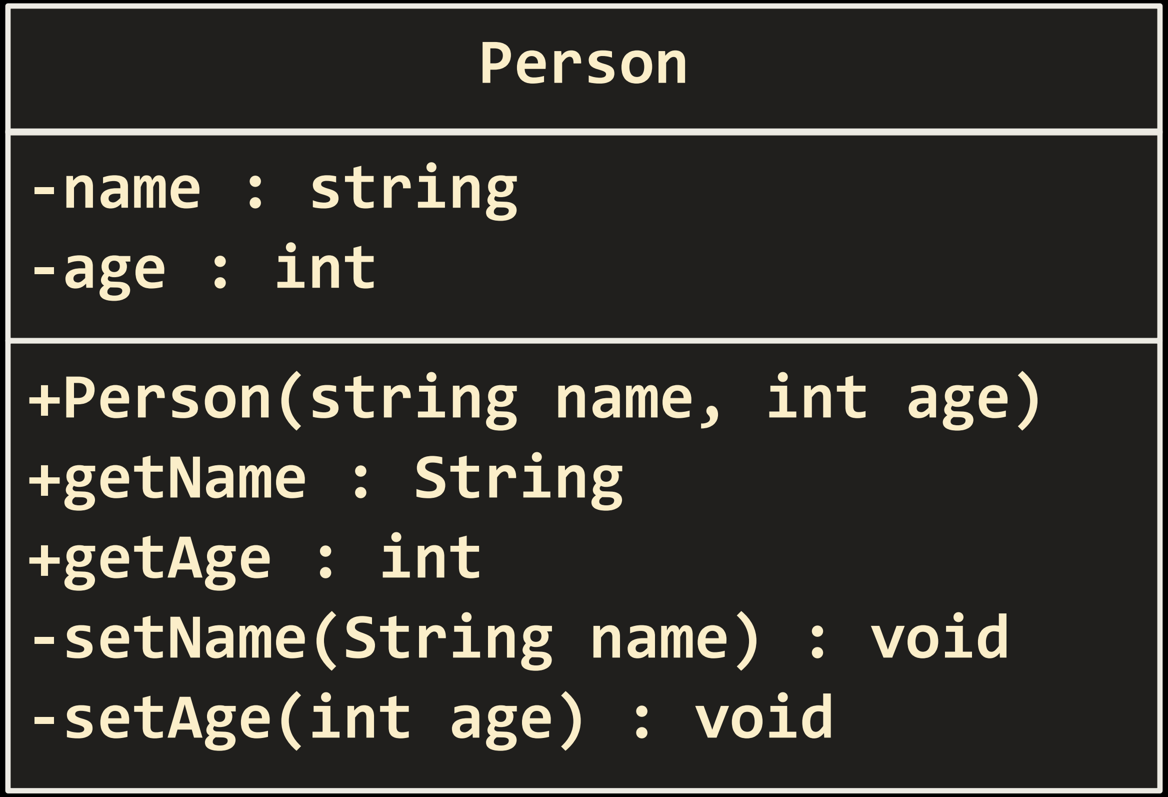
```
class Person {
  private int age;
}
```

- Use getters and setters for data access

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

Encapsulation – Example

Fields should be **private**



```
Person

-name : string
-age : int

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

Keyword this

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

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

- this can invoke current class method

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

Keyword this (2)

- this can invoke current class constructor

```
public Person(String firstName, String lastName) {
  this.firstName = firstName;
  this.lastName = lastName;
}
public Person (String fname, String lName, Integer age) {
  this(fName, lName);
  this.age = age;
}
```

- this can be pass like argument in method or constructor call
- this can be returned from method

Access Modifiers

- Visibility of Class Members

Private Access Modifier

- Main way that an object encapsulates itself and hides data from the outside world

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

- Class and interfaces cannot be private
- Can only be accessed within the declared class itself

Protected Access Modifier

- Can be accessed only by the subclasses in other package

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

- Protected access modifier cannot be applied to class and interfaces
- Preventing 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 rm = new Team("Real");
rm.setName("Real Madrid");
System.out.println(rm.getName());
//Real Madrid
```

Public Access Modifier

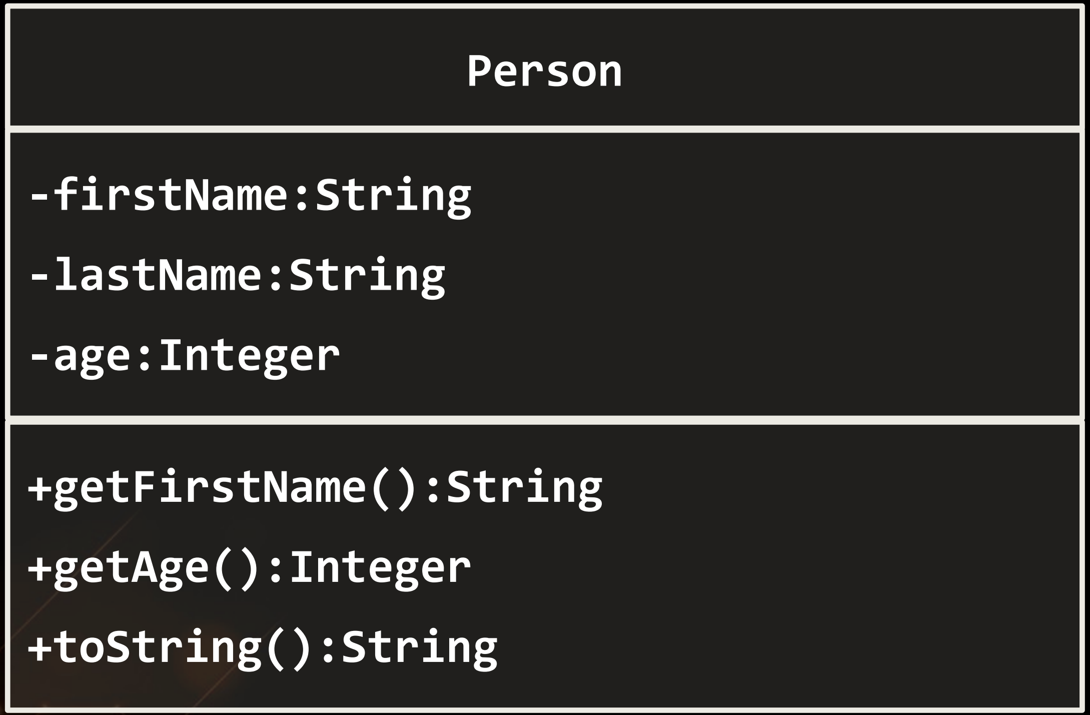
- A Class, method, constructor declared inside a public class can be accessed from any class belonging to the Java Universe

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

- Imports are needed if we try to access public class in different package
- The `main()` method of an application has to be public

Problem: Sort Persons by Name and Age

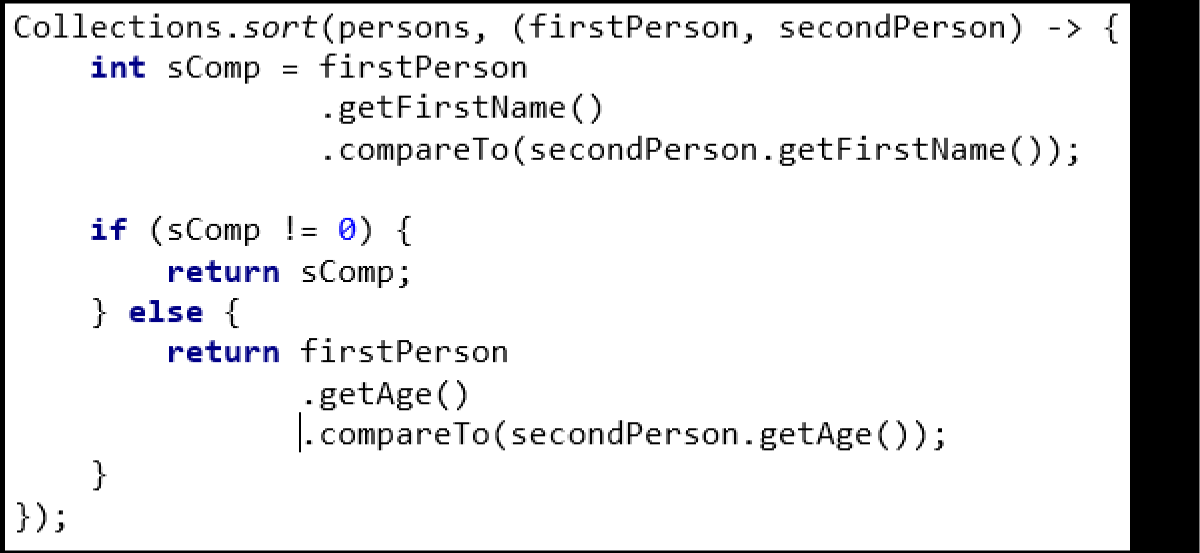
- Create a class Person



Person

-firstName:String
-lastName:String
-age:Integer

+getFirstName():String
+getAge():Integer
+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());
    }
});
```

Solution: Getters and Setters

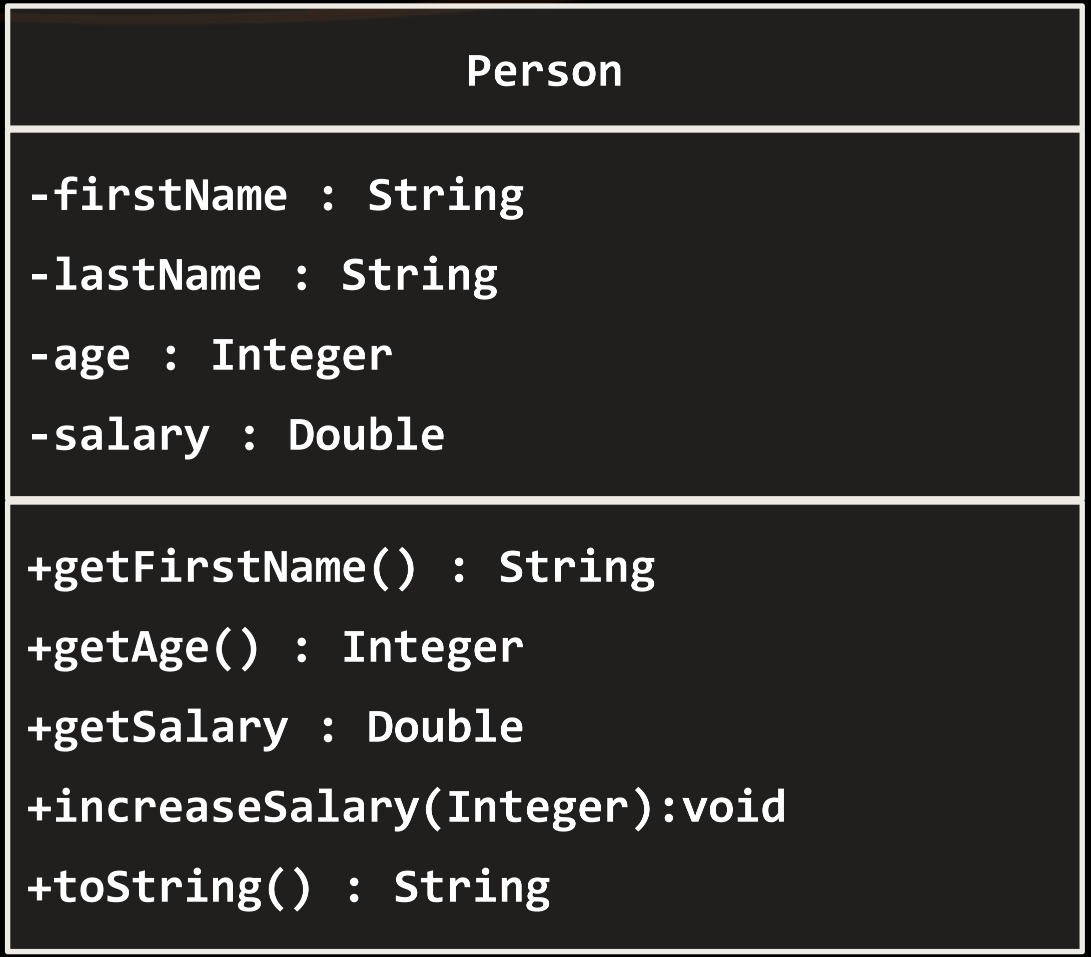
```
public class Person {
  private String firstName;
  private String lastName;
  private Integer age;

  public String getFirstName() {
    return this.firstName; }
  public Integer getAge() {
    return age; }
@Override
  public String toString() { TODO: Add logic}
}
```

Check your solution here: https://judge.softuni.bg/Contests/Practice/Index/475#0

Problem: Salary Increase

- Expand Person with salary
- Add getter for salary
- Add method, which update salary with given percent
- Persons younger than 30 get half increase than normal



```
Person

-firstName : String
-lastName : String
-age : Integer
-salary : Double

+getFirstName() : String
+getAge() : Integer
+getSalary : Double
+increaseSalary(Integer):void
+toString() : String
```

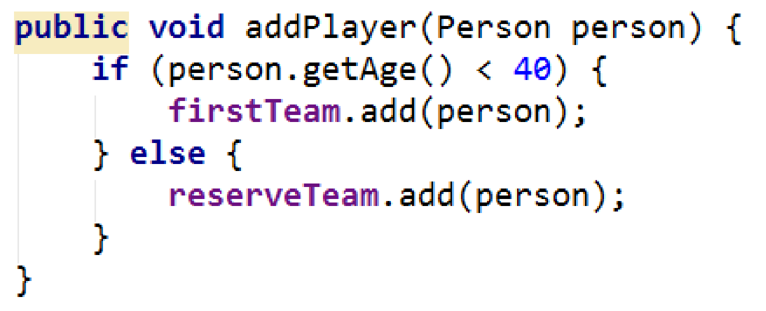
Solution: Getters and Setters

- Expand Person from previous task

```
public class Person {
  private Double salary;

  public String getSalary() { return this.salary; }
  public void increaseSalary(Integer percentBonus) {
    if (this.age > 30) {
      this.salary += this.salary * bonus / 100;
    } else {
      this.salary += this.salary * bonus / 200;
    }
  } }
```

Check your solution here: https://judge.softuni.bg/Contests/Practice/Index/475#0



```java
public void addPlayer(Person person) {
    if (person.getAge() < 40) {
        firstTeam.add(person);
    } else {
        reserveTeam.add(person);
    }
}
```

Exercises in Class

- Implement Getters and Setters

Encapsulation in Java

Validation

- Data validation happen in setters

```
private void setSalary(Double salary) {
  if (salary < 460) {
    throw new IllegalArgumentException
             ("Salary cannot be less than 460 leva");
  }
    this.salary = salary;
}
```

Better throw exception, than print to Console

- Don’t couple your class with Console
- Contributor of your class have to think about handle Exceptions

Validation (2)

- Constructors use private setter with validation logic

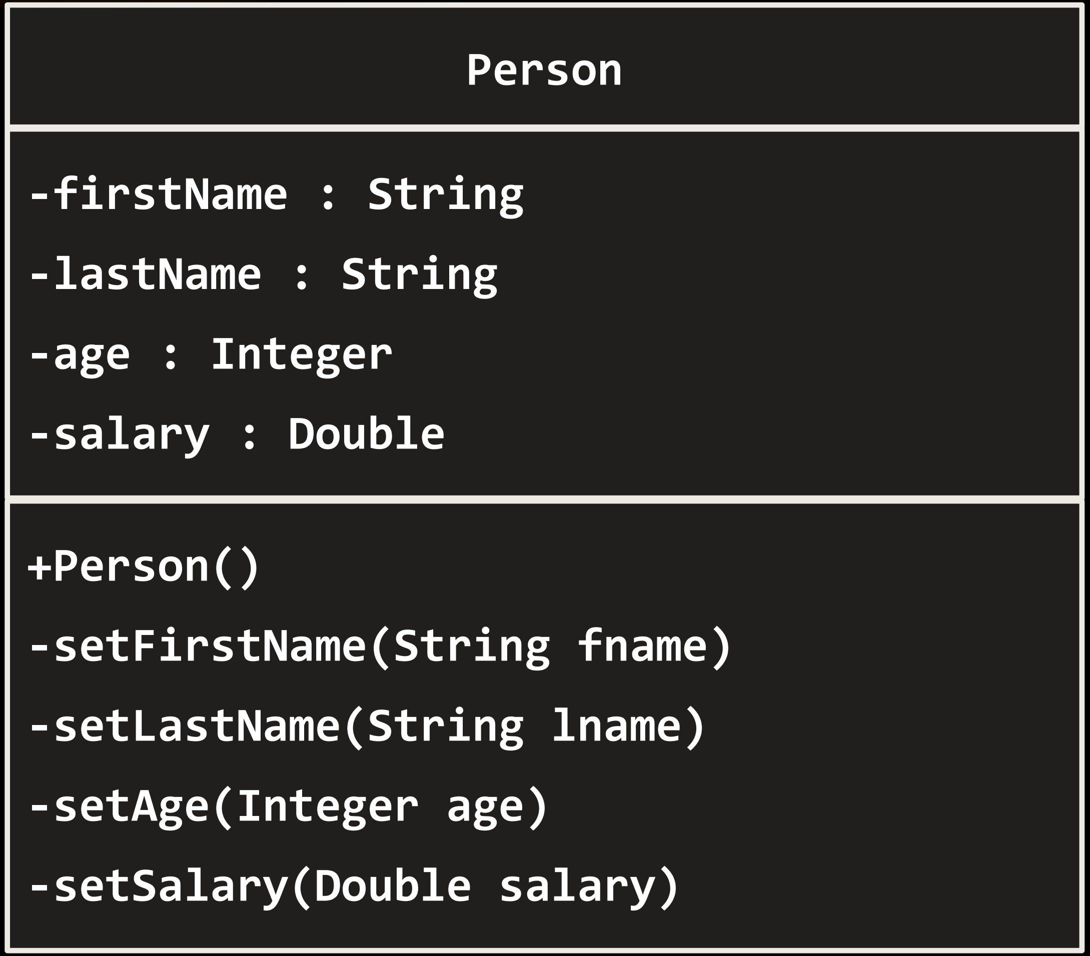
```
public Person(String firstName, String lastName,
              Integer age, Double salary) {
  setFirstName(firstName);
  setLastName(lastName);
  setAge(age);
  setSalary(salary);
}
```

Validation is happen inside of setter

- Guarantee valid state of object in its creation
- Guarantee valid state for public setters

Problem: Validate Data

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



```
Person
-firstName : String
-lastName : String
-age : Integer
-salary : Double
+Person()
-setFirstName(String fname)
-setLastName(String lname)
-setAge(Integer age)
-setSalary(Double salary)
```

Solution: Validate Data

```
TODO: Add validation for firstName
TODO: Add validation for lastName
private void setAge(Integer 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/Practice/Index/475#0

Immutable Objects

- When you have a reference to an instance of an object, the contents of that instance **cannot** be altered

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

```
old String
old String
```

Mutable Objects

- When you have a reference to an instance of an object, the contents of that instance **can** be altered

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

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

Mutable Fields

- private mutable fields are still don’t encapsulated

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

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

- In this case getter is setter too

Mutable Fields (2)

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

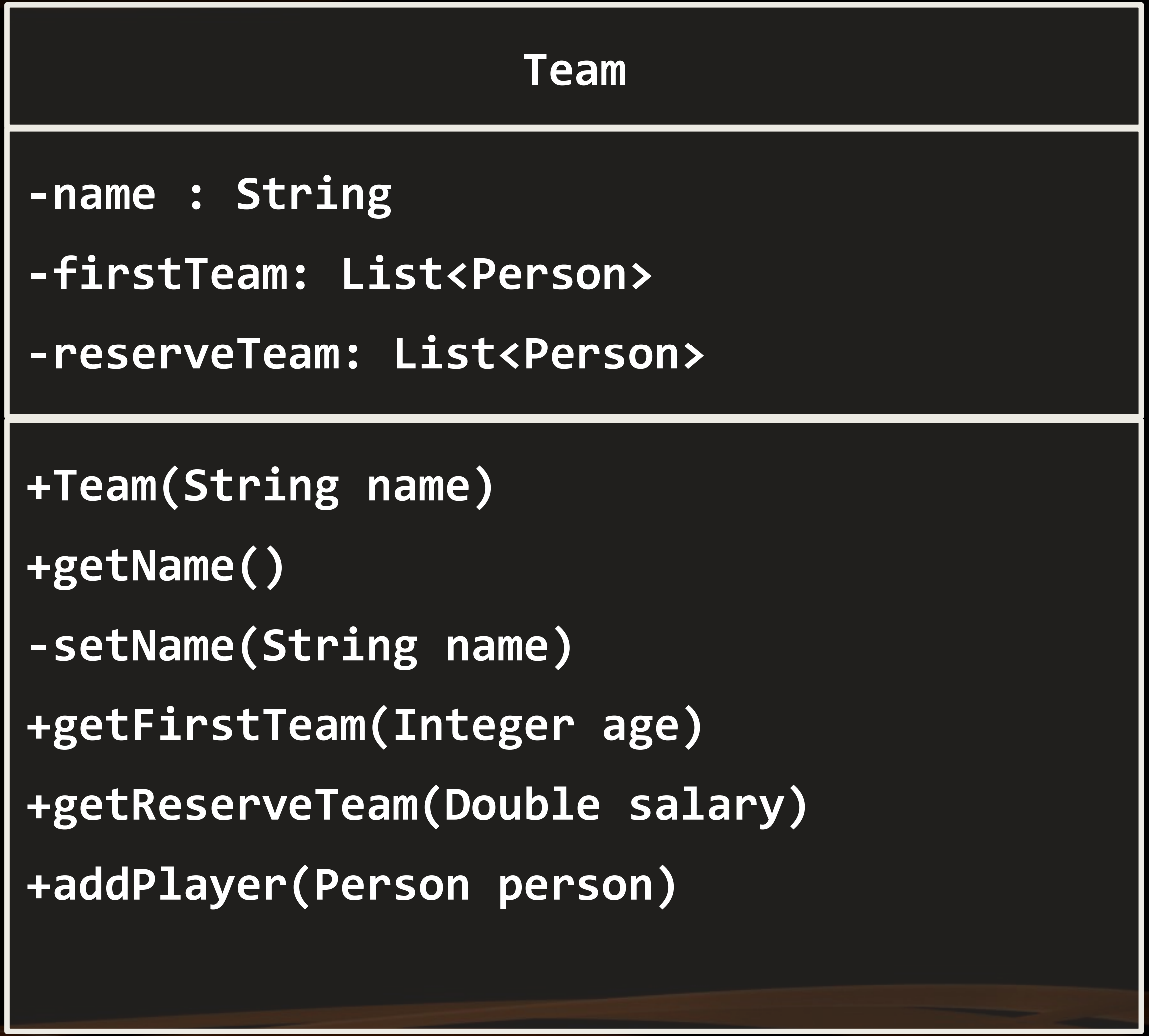
```
class Team {
  private List<Person> players;
  public addPlayer(Person person) {
    this.players.add(person);
  }
  public List<Person> getPlayers() {
    return Collections.unmodifiableList(players);
  }
```

Add new methods for functionality over list

Return 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



```
Team

-name : String
-firstTeam: List<Person>
-reserveTeam: List<Person>

+Team(String name)
+getName()
-setName(String name)
+getFirstTeam(Integer age)
+getReserveTeam(Double salary)
+addPlayer(Person person)
```

Solution: Validate Data

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

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

Check your solution here: https://judge.softuni.bg/Contests/Practice/Index/475#0

Keyword final

- final class can't be extended

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

- final method can't be overridden

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

Keyword final (2)

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

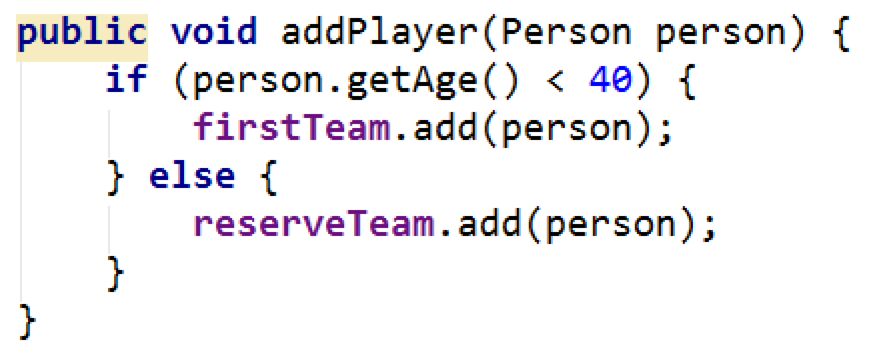
```
Private final String name;
Private final List<Person> firstTeam;

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

Compile time error

Encapsulation – Benefits

- Reduces complexity
- Structural changes remain local
- Allows validations and data binding



```
public void addPlayer(Person person) {
    if (person.getAge() < 40) {
        firstTeam.add(person);
    } else {
        reserveTeam.add(person);
    }
}
```

Exercises in Class

- Validations, Mutable and Immutable Objects

Summary

- Encapsulation hides implementation
- Access modifiers
- Encapsulation reduces complexity
- Ensures that structural changes remain local
- Mutable objects
- Immutable objects

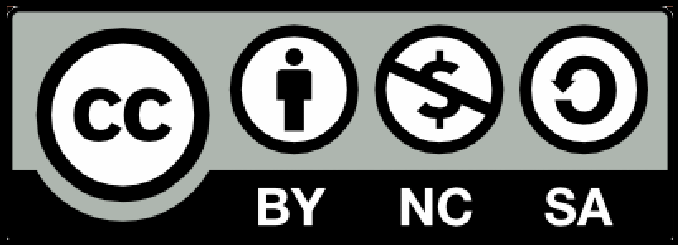
Java Syntax

Questions?

- https://softuni.bg/courses/java-fundamentals

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
  - "Fundamentals of Computer Programming with Java" book by Svetlin Nakov & Co. under CC-BY-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 @ YouTube
  - youtube.com/SoftwareUniversity
- Software University Forums – forum.softuni.bg