softserve experience matters

Java Core

Class and Object

IT Academy
04/2016

Agenda

- Class and Object
- Access to data
- Fields of class
- Getters and Setters
- Constructors
- Methods of class
- Creating objects
- Examples

Class and Object

A *class* is a prototype (template) from which objects are created

An *object* is a software bundle of related state and behavior

Student

has

- Last name
- First name
- Age
- List of courses

can

- Pass an exam
- Enroll to course

student1

- Last name - Petrenko
- First name - Ostap
- Age - 19
- List of courses – Java, MQC

student2

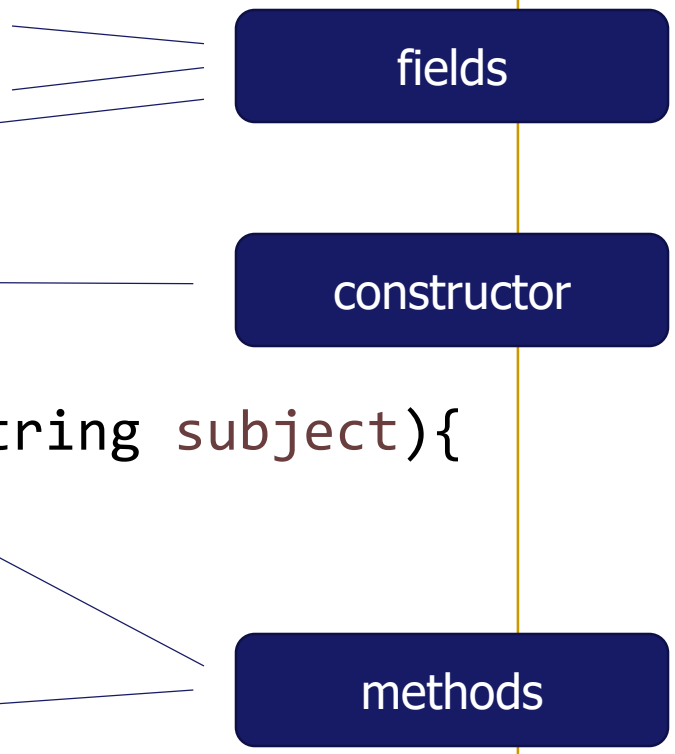
- Last name - Romaniv
- First name - Maryna
- Age - 21
- List of courses – Java, ATQC

Class

```
<access specifier> class ClassName {
        // fields
        <access specifier> <data type> variable1;
        ...
        <access specifier> <data type> variableN;

        // constructors
        <access specifier> ClassName(parameter_list1){
            // method body    }
        ...
         <access specifier> ClassName(parameter_listN){
            // method body    }
        // methods
        <access specifier> <return type> method1(parameter_list){
            // method body    }
        ...
         <access specifier> <return type> methodN(parameter_list){
            // method body    }
```

Class

```
public class Student {
    private String lastName;
    private String firstName;
    private int age;

    private Student(){}

    public boolean passExam(String subject){
        //do something
        return true;
    }
    public void print(){
        //do something
    }
}
```



Access to data

```
public class Student {...}
private int age;
public void print(){}
```

access specifier

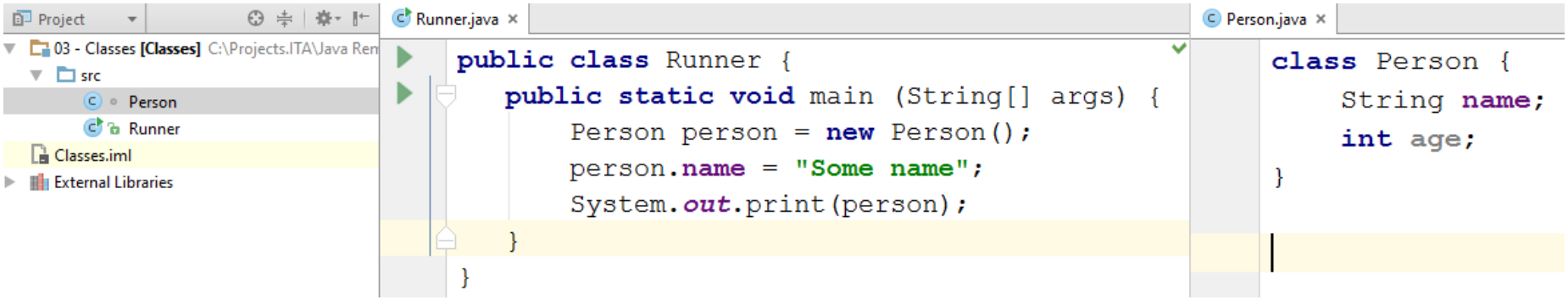
data type

Controlling Access to Members of a Class

	Class	Package	Subclass	World
private	Y	—	—	—
(not)	Y	Y	—	—
protected	Y	Y	Y	—
public	Y	Y	Y	Y

Special Requirements to source files

- a source code file (.java) can have only *one public class*
- name of this class should be *exactly the same* of file name before extension (including casing)
- source code file can have *any number* of non-public classes
- most code conventions require use only *one top-level class* per file



Default values for fields

Type	Bits	Value	
byte	8	-128 < x < 127	0
short	16	-32768 < x < 32767	0
int	32	-2147483648 < x < 2147483647	0
long	64	-9223720368547758O8 < x < 9223720368547758O7	0L
char	16	0 < x < 65536	'\u0000'
float	32	3,4e-38 < |x| < 3,4e38; 7-8 digits	0.0f
double	64	1,7e-308 < |x| < 1,7e308; 17 digits	0.0d
boolean	8	false, true	false
String	variable	Symbols sequence of Unicode characters.	null
Object	variable	Any object	null

Type casting

Widening (implicit or automatic) type casting take place when, the two types are compatible the target type is larger than the source type


```
int i = 100;
long l = i;       //no explicit type casting required
float f = l;      //no explicit type casting required
```

When you are assigning a larger type value to a variable of smaller type, then you need to perform ***narrowing*** (explicit) type casting.


```
double d = 100.04;
long l = (long) d; //explicit type casting required
int i = (int) l;   //explicit type casting required
```

Methods and overloading

- Methods are functions that are executed in context of object
- Always have full access to data of object
- Object can have multiple methods with same name but different signature (type and order of parameters)
- Signature doesn't include return type, methods can't be overloaded by return types

```
class Person {
    String name;
    public void print() {
        System.out.println(name);
    }
    public void print(String s) {
        System.out.println(s + " " + name);
    }
}
```

Variable length arguments

- Methods in Java support arguments of variable length
-

```
public class Util {
    public static void print (String welcomeMessage,
                Object... messages) {
        System.out.print(welcomeMessage);
        for (Object msg: messages) {
            System.out.print(msg);
        }
    }
}

public class Runner {
   public static void main (String[] args) {
        Person person = new Person();
        Util.print("Any ", "argment ", "possible",
                10, 20.5, false, person);
    }
}
```

Access to fields

The following class uses public access control:

```
public class Student {
  public String name;
  public int age;
  ...
}
```

```
Student stud = new Student();
stud.name = “Krystyna”;
stud.age = 22;
```

Do not make so!

Getters and Setters

The following class uses private access control:

```
public class Student {
    private String name;

    public String getName() {
        return this.name;
    }
     public void setName(String name) {
        this.name = name;
    }
}
```

Getters and Setters

```
Student student = new Student();
```


```
student.setName("Franko");
```


```
String nameStud =
            student.getName();
```

Getters and Setters can be Complex

```
public class Sum {
    private int a, b, c;

    void setA(int m) { this.a = m; c = a + b; }
    void setB(int n) { this.b = n; c = a + b; }

    int getA() { return this.a; }
    int getB() { return this.b; }
    int getC() { return this.c; }

    public void sum(int m, int n) {
        this.a = m; this.b = n;
        this.c = m + n;
    }
}
```

Keyword "this"

- **this** always points to current object
- can't lose context like JavaScript
- not required in most cases
- often needed to distinguish between parameters and fields:

```
public class SomeClass {
    private int a;

    void setA(int a) {this.a = a;}

}
```

Keyword 'static'

- Keyword 'static' indicates that some class member (method or field) is not associated with any particular object
- Static members should be accessible by class name (good practice, not required by language itself)

```
public class Helper {
    private static String message;
    public static void setMessage(String message) {
        Helper.message = message;
    }

    public static void print() {
        System.out.println(message);
    }
}
```

Keyword 'static'

```
public class Runner {

    public static void main (String[] args) {

        Helper.setMessage("hello");
        Helper.print();

       // Not recommended:
        Helper helper = new Helper();
        helper.setMessage("new message");
        helper.print();
    }

}
```

Constructors

- Constructors – special kind of methods called when instance created
- Name should be same as a class
- Class may have multiple overloaded constructors
- If not provided any constructor, Java provides default parameterless empty constructor

```
public class Person {
    private String name;
    private int age;

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

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

    public String getName() {
        return name;
    }

    public int getAge() {
        return age;
    }
}
```

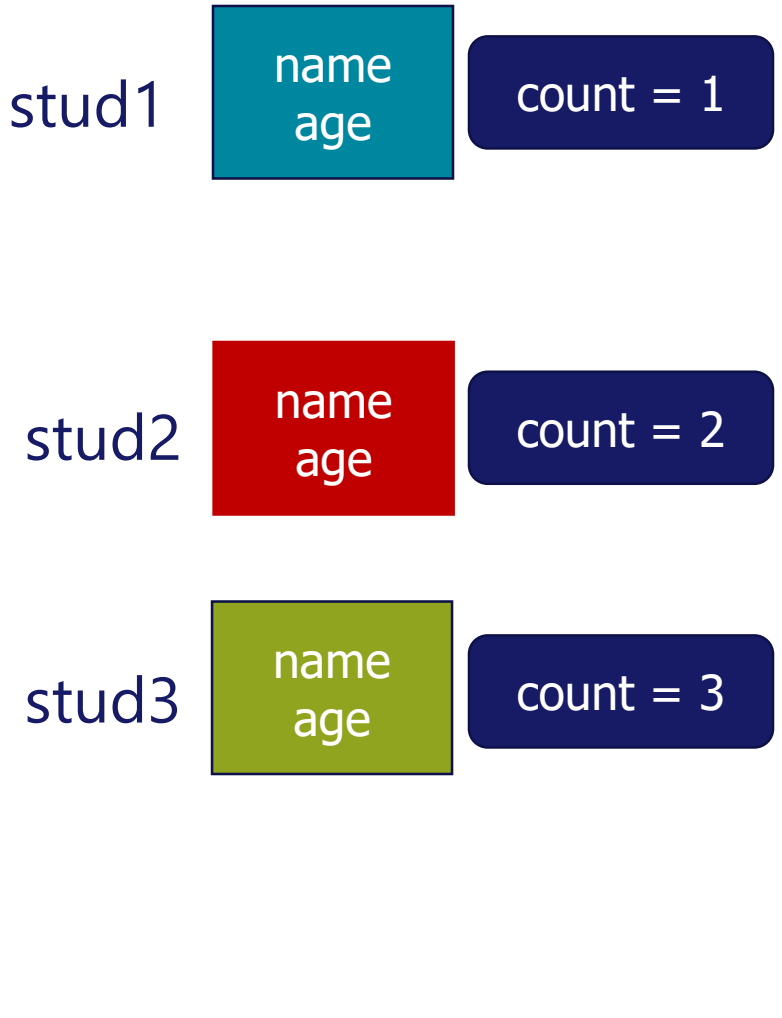
Constructors

```
public class Student {
    private String name;
    private int age;

    public static int count = 0;

    public Student(){count++;}

    public Student(String name){
        this.name = name;
        count++;
    }

    public Student(String name, int age){
        this.name = name;
        this.age = age;
        count++;
    }  ... getters, setters and methods
}
```

They have the same name

Creating objects – new()

```
Student stud1 = new Student();
  stud1.setName("Dmytro");
  stud1.setAge(25);

Student stud2 =
     new Student("Olga");
 stud2.setAge(24);

Student stud3 =
     new Student("Ivan", 26);

int n = Student.count;
```



Private constructor

- Making constructor private will prevent creating instances of a class from other classes
- Still allows creating instances inside static methods of the class

```
public class Helper {
    private Helper () {}

    private static String message;

    public static void setMessage(String message) {
        Helper.message = message;
    }

    public static void print() {
        System.out.println(message);
    }

    public static Helper getHelper() {
        return new Helper();
    }
}
```

```
public class Runner {
   public static void main (String[] args) {
        Helper.setMessage("hello");
        Helper.print();
        // Not recommended:
        //! Helper helper = new Helper();
        Helper helper = Helper.getHelper();
        helper.setMessage("new message");
        helper.print();
    }
}
```

toString()

```
System.out.println(student);
```

com.edu.Student@659e0bfd

```
@Override
public String toString() {
      return "Student
       [lastNname=" + lastNname +
         ", firstName=" + firstName +
         ", age=" + age + "]";
}
```

Student [lastNname=Ivanov, firstName=Vasiy, age=22]

Example

Create Console Application project in Java.
Add class **Student** to the project.
Class Student should consists of

a) two private fields: name and rating;
b) properties for access to these fields
c) static field avgRating – average rating of all students
d) default constructor and constructor with parameters
e) methods:
   - *betterStudent* - to definite the better student (between two, return true or false)
   - *toString* - to output information about student
   - *changeRating* - to change the rating of student

In the method main() create 3 objects of Student type and input information about them.
Display the average and total rating of all student.

Practical task

Create Console Application project in Java.
Add class **Employee** to the project.
Class Employee should consists of

a) three private fields: name, rate and hours;
b) static field totalSum
c) properties for access to these fields;
d) default constructor, constructor with 2 parameters (name and rate) and constructor with 3 parameters;
e) methods:
   - *salary* - to calculate the salary of person (rate * hours)
   - *toString* - to output information about employee
   - *changeRate* - to change the rate of employee
   - *bonuses* – to calculate 10% from salary

In the method main() create 3 objects of Employee type. Input information about them.
Display the total salary of all workers to screen

Homework

Create Console Application project in Java.

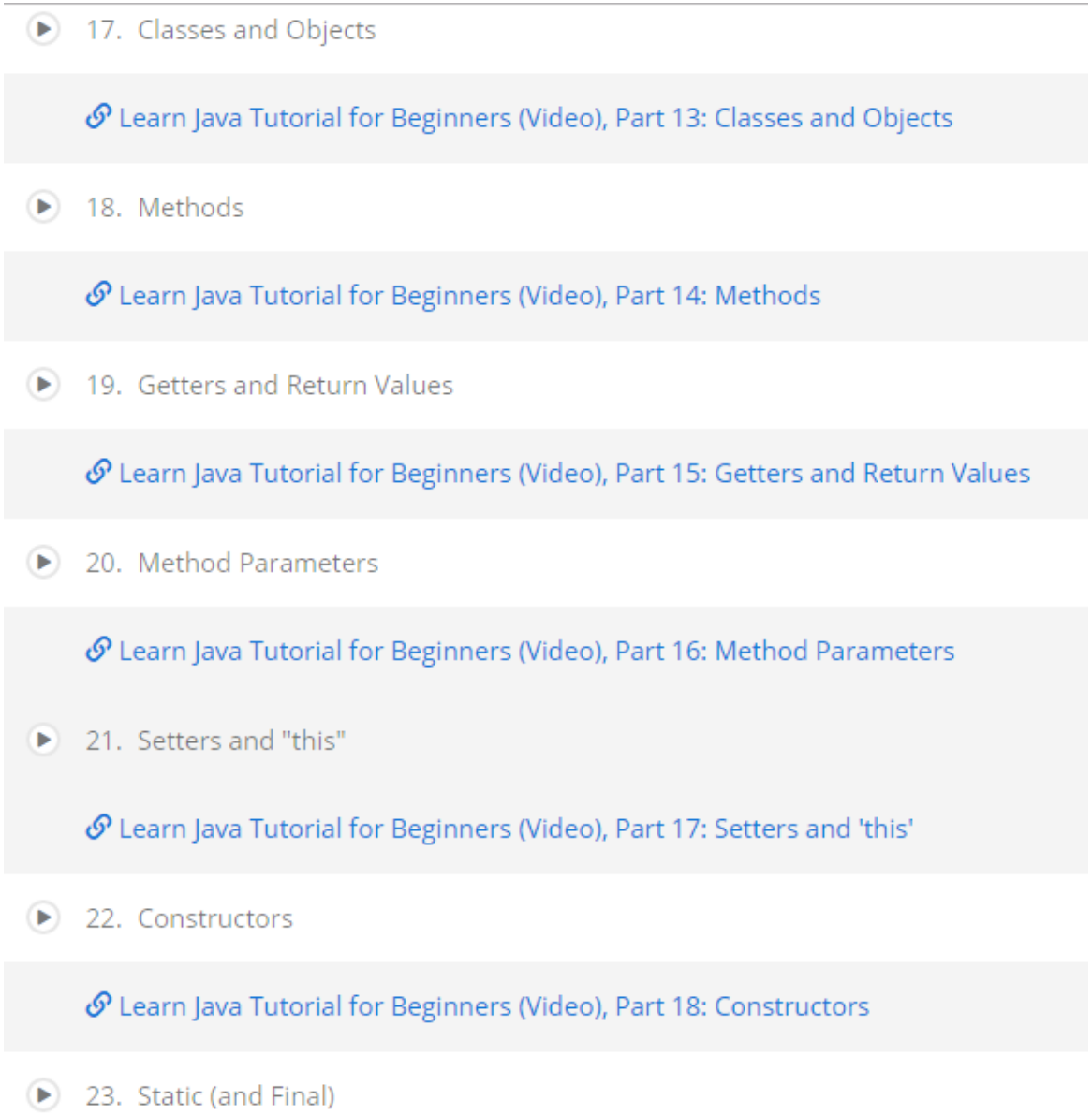
Add class **Person** to the project.

Class Person should consists of

a) two private fields: name and birthYear (the birthday year)
b) properties for access to these fields
c) default constructor and constructor with 2 parameters
d) methods:
   - *age* - to calculate the age of person
   - *input* - to input information about person
   - *output* - to output information about person
   - *changeName* - to change the name of person

In the method main() create 5 objects of Person type and input information about them.

- UDEMY course "Java Tutorial for Complete Beginners": https://www.udemy.com/java-tutorial/
- Complete lessons 17-23:



softserve experience matters

The end

USA HQ
Toll Free: 866-687-3588
Tel: +1-512-516-8880

Ukraine HQ
Tel: +380-32-240-9090

Bulgaria
Tel: +359-2-902-3760

Germany
Tel: +49-69-2602-5857

Netherlands
Tel: +31-20-262-33-23

Poland
Tel: +48-71-382-2800

UK
Tel: +44-207-544-8414

EMAIL
info@softserveinc.com

WEBSITE:
www.softserveinc.com