INNOPOLIS UNIVERSITY

Object-oriented programming in Java

Stanislav I. Protasov

27/07/2015

Agenda

- Program
  - Imperative vs declarative
- Object
  - State, behavior, identity
  - Class and object in Java. Fields, methods, scope
  - Reference type
- “3 Whales” of OOP
  - Encapsulation
  - Inheritance
  - Polymorphism
- Special cases
  - Interfaces
  - Static methods and fields
  - enums
  - Boxing

Imperative and declarative ways

VS

Coding approaches

These are imperative programs:

```
.MODEL SMALL
.STACK 100h
.DATA
    HelloMessage DB 'Hello World',13,10,'$'
.CODE
START:
    mov ax,@data
    mov ds,ax
    mov ah,9
    mov dx,OFFSET HelloMessage
    int 21h
    mov ah,4ch
    int 21h
END START
```

```
def greet(name):
    print 'Hello', name
greet('Jack')
greet('Jill')
greet('Bob')
```

This is declarative program:

```
(defrule can-take-201 "Eligible for 201?"
   (has-taken (ID ?num) (course MATH221))
   =>
   (assert (can-take (ID ?num) (course CSCI201))))

(defrule can-take-301 "Eligible for 301?"
   (has-taken (ID ?num) (course CSCI201))
   (has-taken (ID ?num) (course MATH273))
   =>
   (assert (can-take (ID ?num) (course CSCI301))))

(defrule can-take-273 "Eligible for 273?"
   (has-taken (ID ?num) (course MATH221))
   =>
   (assert (can-take (ID ?num) (course CSCI273))))

   ...............

(deffacts Names "Associates names with their IDs"
   (who-is-it (name Sasha Pankratov)
              (ID 00384175)))

(deffacts Sasha-Pankratov "Courses taken"
   (has-taken (ID 00384175)
              (course MATH221)))
```

OOP as a part of evolutional chain

Approach	Who to order	What to order
Machine codes	CPU	Execute instruction
Procedural languages	Procedure	Execute something logically linked
OO languages	**Object**	**To behave as expected**

Class and Object

Prototype-based OOP (JavaScript)

Object: state, behavior, identity

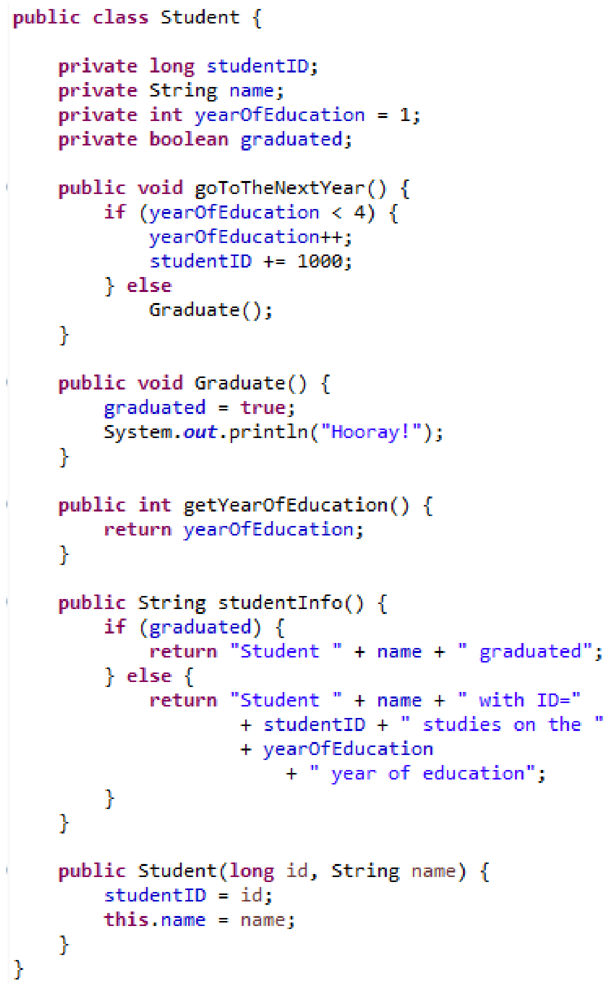
Objects can be treated as a **model** of real (or unreal) world things.

OBJECT = STATE + BEHAVIOR + IDENTITY

- Human's models:
  - Citizen
    (passport data, elections and traffic law, ID)
  - Family member
    (relations, respect and traditions, name/nickname)
  - Student
    (grade and marks, studying, Student ID)

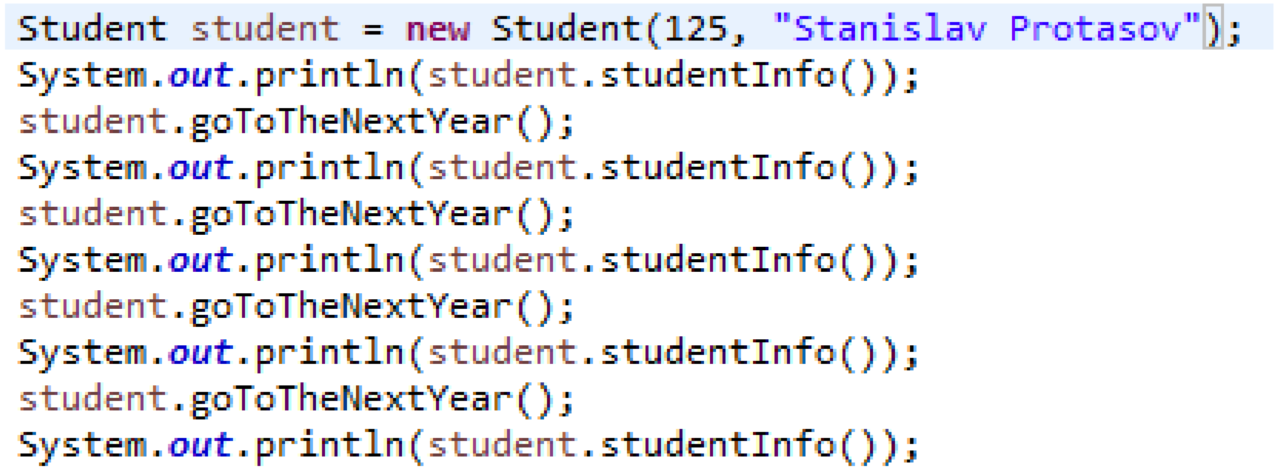
Declaring class in Java; scopes

CLASS:



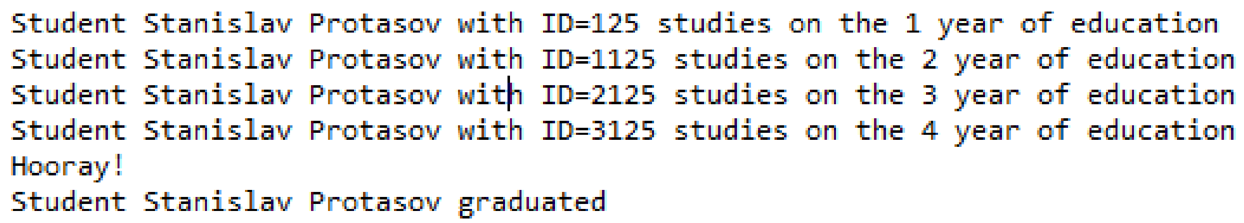
```java
public class Student {

    private long studentID;
    private String name;
    private int yearOfEducation = 1;
    private boolean graduated;

    public void goToTheNextYear() {
        if (yearOfEducation < 4) {
            yearOfEducation++;
            studentID += 1000;
        } else
            Graduate();
    }

    public void Graduate() {
        graduated = true;
        System.out.println("Hooray!");
    }

    public int getYearOfEducation() {
        return yearOfEducation;
    }

    public String studentInfo() {
        if (graduated) {
            return "Student " + name + " graduated";
        } else {
            return "Student " + name + " with ID="
                    + studentID + " studies on the "
                    + yearOfEducation
                        + " year of education";
        }
    }

    public Student(long id, String name) {
        studentID = id;
        this.name = name;
    }
}
```

ORDERS:



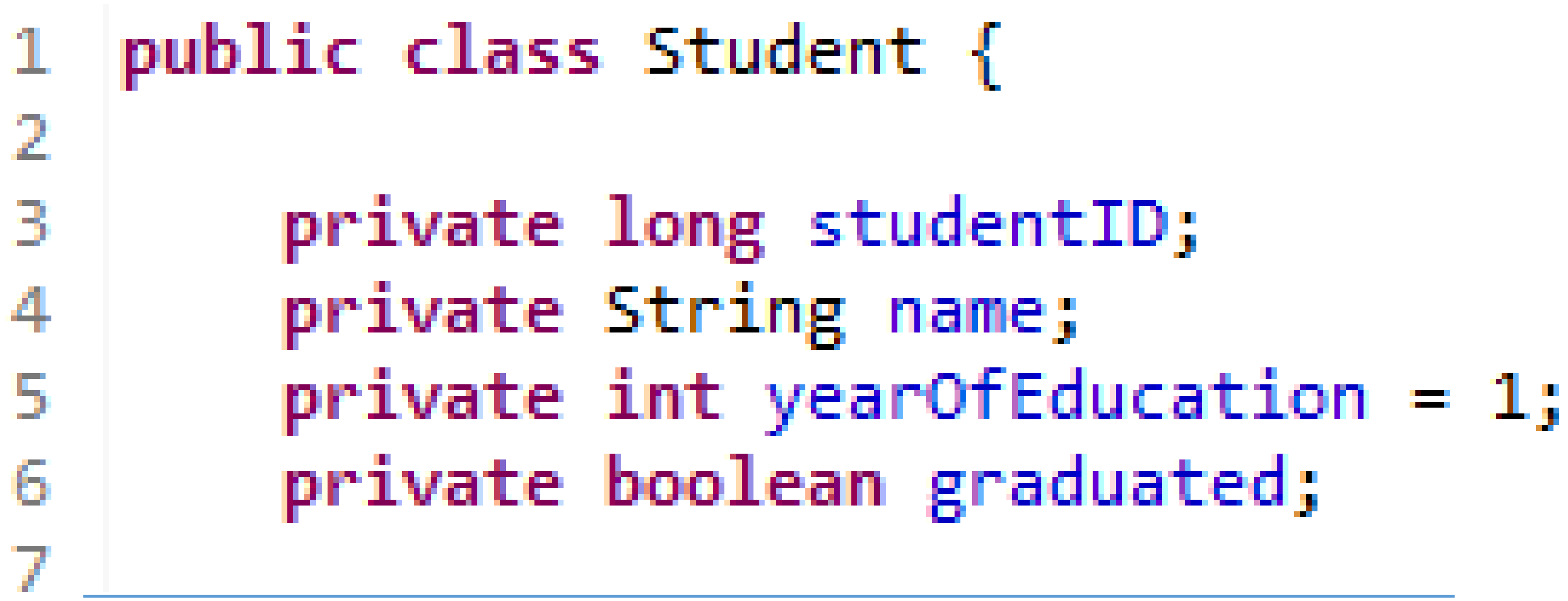
```java
Student student = new Student(125, "Stanislav Protasov");
System.out.println(student.studentInfo());
student.goToTheNextYear();
System.out.println(student.studentInfo());
student.goToTheNextYear();
System.out.println(student.studentInfo());
student.goToTheNextYear();
System.out.println(student.studentInfo());
student.goToTheNextYear();
System.out.println(student.studentInfo());
```

OUTPUT:

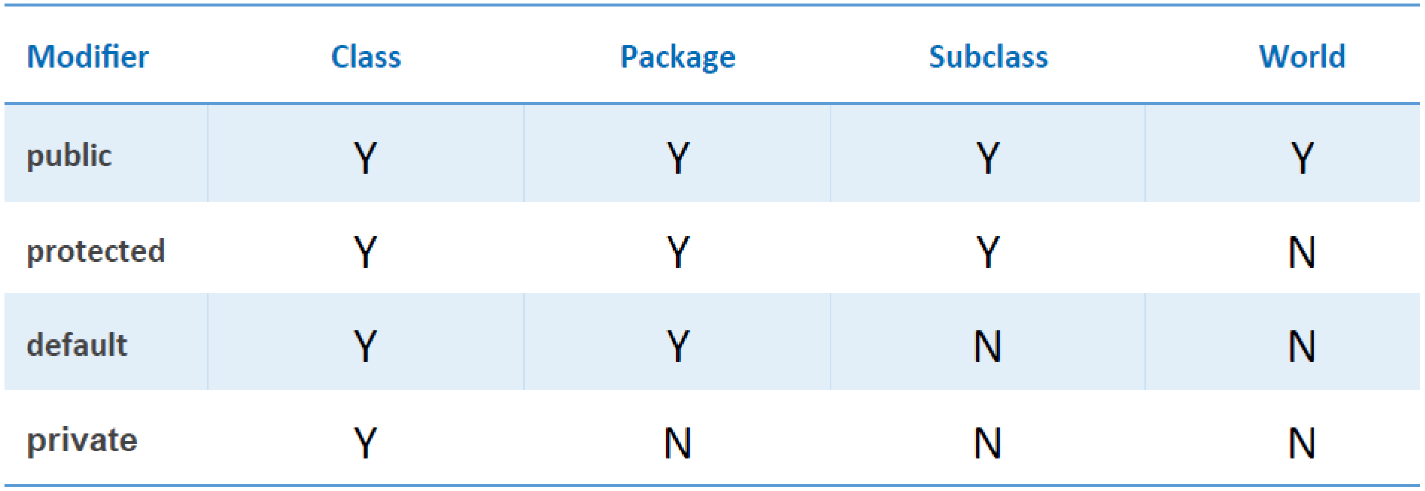


```
Student Stanislav Protasov with ID=125 studies on the 1 year of education
Student Stanislav Protasov with ID=1125 studies on the 2 year of education
Student Stanislav Protasov with ID=2125 studies on the 3 year of education
Student Stanislav Protasov with ID=3125 studies on the 4 year of education
Hooray!
Student Stanislav Protasov graduated
```

Header and fields

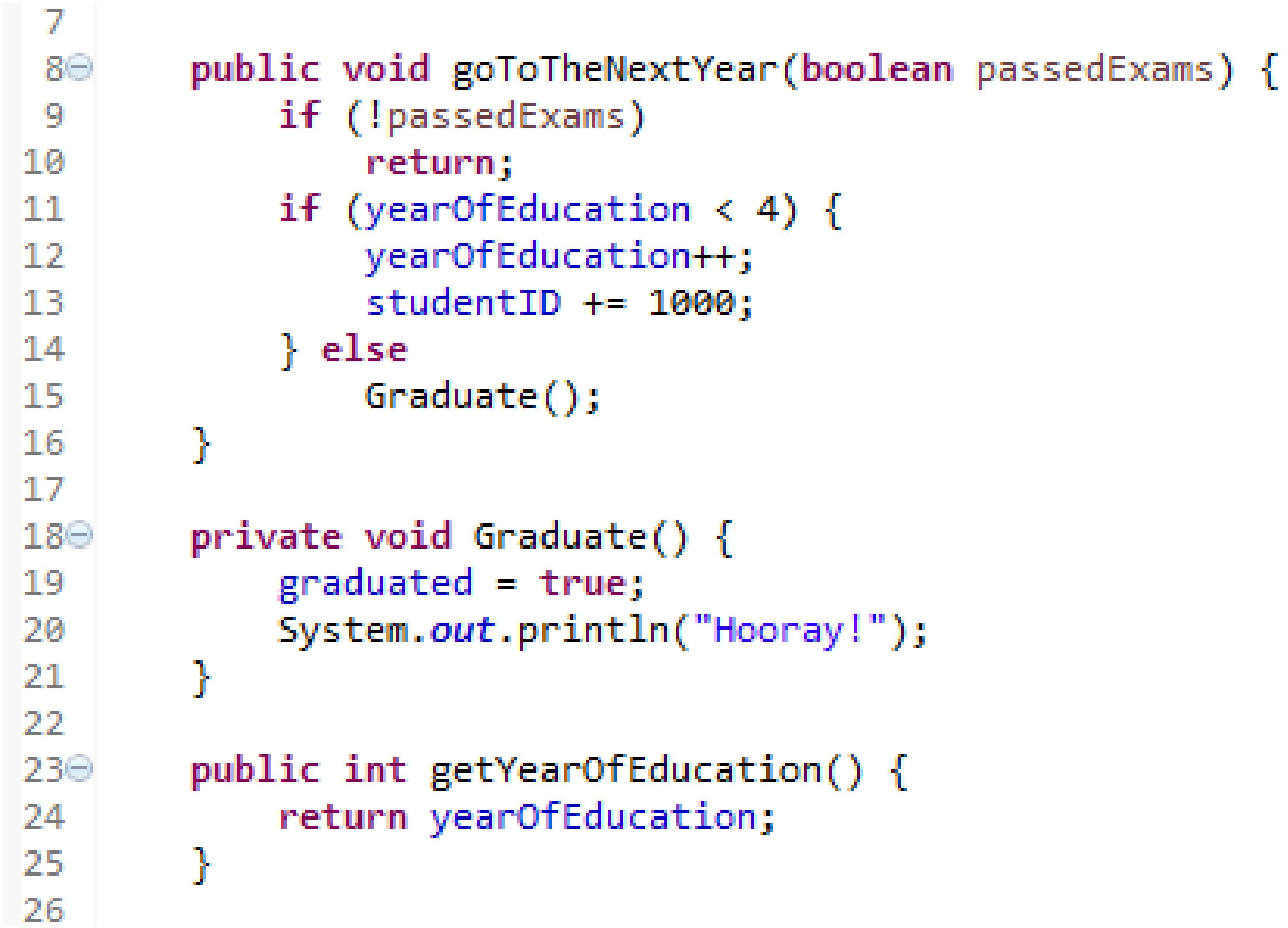


```
public class Student {

    private long studentID;
    private String name;
    private int yearOfEducation = 1;
    private boolean graduated;
```



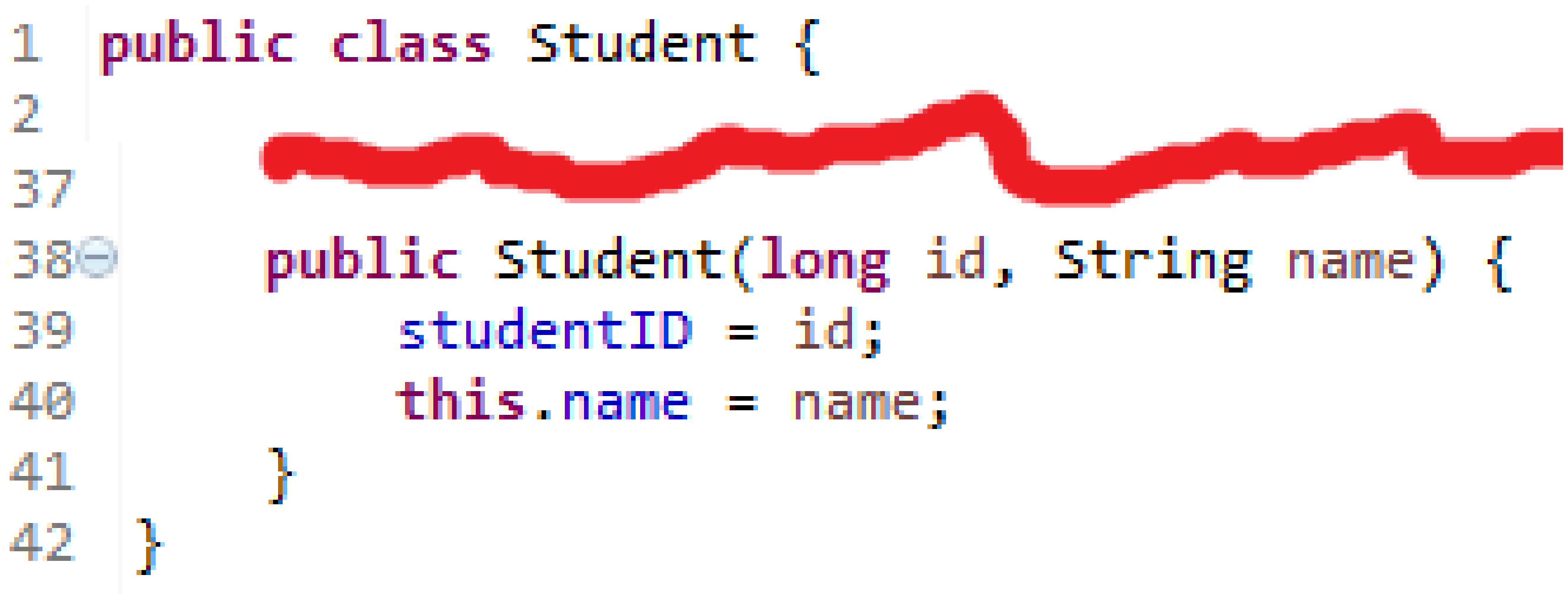
Modifier	Class	Package	Subclass	World
public	Y	Y	Y	Y
protected	Y	Y	Y	N
default	Y	Y	N	N
private	Y	N	N	N

Methods



```java
    public void goToTheNextYear(boolean passedExams) {
        if (!passedExams)
            return;
        if (yearOfEducation < 4) {
            yearOfEducation++;
            studentID += 1000;
        } else
            Graduate();
    }

    private void Graduate() {
        graduated = true;
        System.out.println("Hooray!");
    }

    public int getYearOfEducation() {
        return yearOfEducation;
    }

```

Constructors



```java
public class Student {


    public Student(long id, String name) {
        studentID = id;
        this.name = name;
    }
}
```

Method overload

```
public void doThings(int param1, int param2) {

}

public int doThings(int param1) {

}
```

Object: reference type

Value (primitive) type variable stores the **value** it was assigned.

Reference type (object) variable stores the **reference** to the place where object’s state is saved.

```
double i = 5; // 8 bytes allocated
double j = i; // 8 more bytes allocated
Object o = new Object(); // 4 bytes + class size
Object p = o;            // only 4 bytes allocated!
```

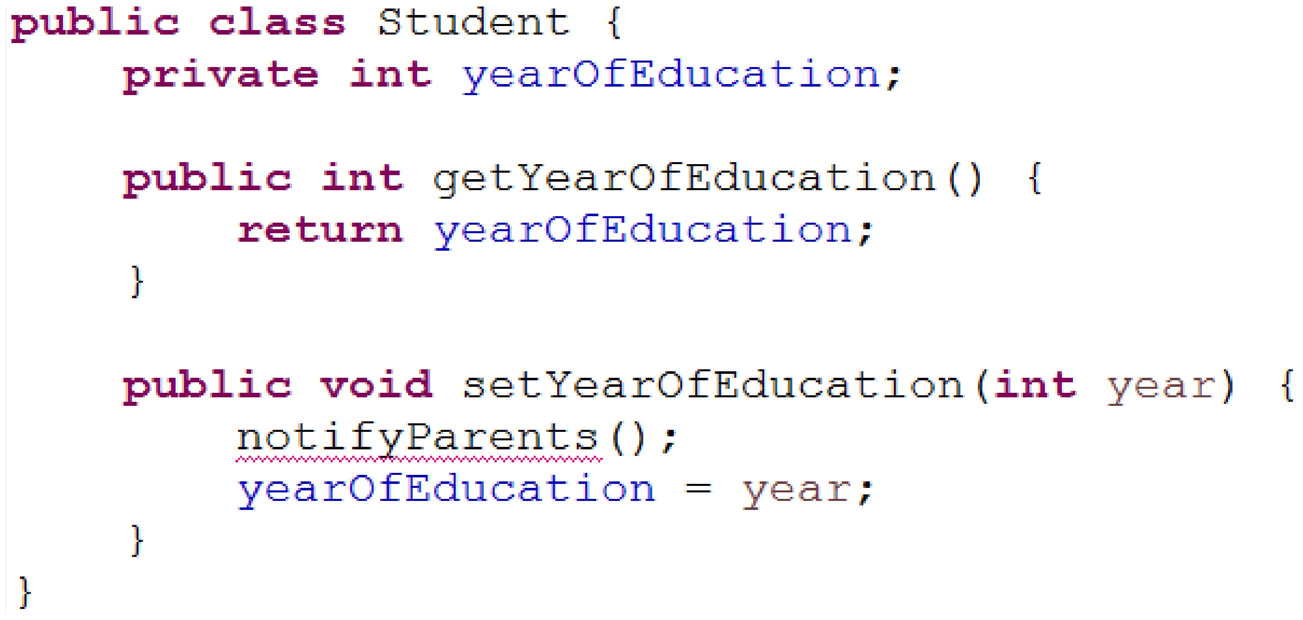
OOP Laws

POLYMORPHISM
ENCAPSULATION
INHERITANCE

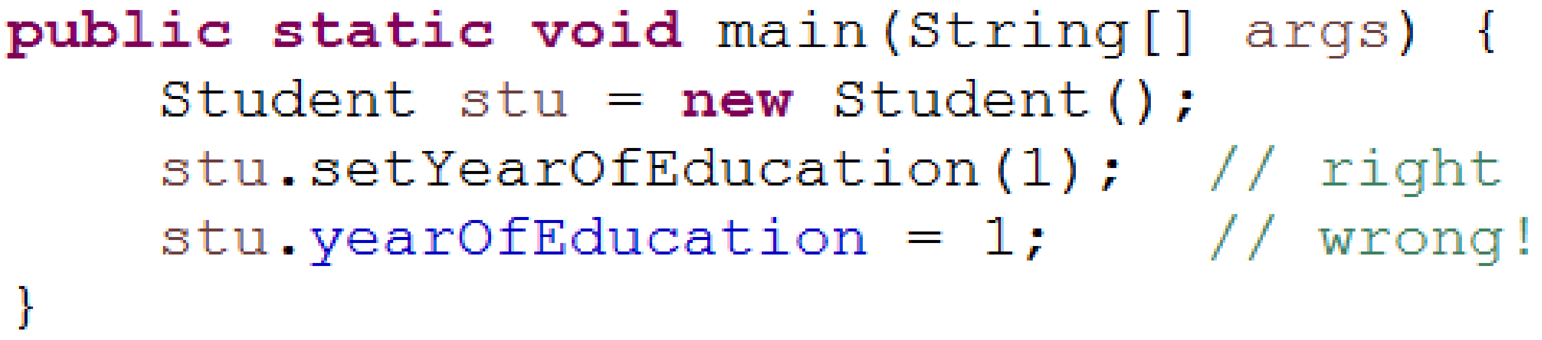
Encapsulation: privacy in OOP world

- Encapsulation:
  - Gathering all information (data and behavior) about some object into a single structure and **considering it as a whole thing**.
  - **Protecting object’s state** from being changed directly by accessing fields

Encapsulation violation



```
public class Student {
    private int yearOfEducation;

    public int getYearOfEducation() {
        return yearOfEducation;
    }

    public void setYearOfEducation(int year) {
        notifyParents();
        yearOfEducation = year;
    }
}
```



```
public static void main(String[] args) {
    Student stu = new Student();
    stu.setYearOfEducation(1);  // right
    stu.yearOfEducation = 1;    // wrong!
}
```

Inheritance: being lazy is not a crime

- Inheritance:
  - Ability to reuse parent class' *state** and *behavior*.

Inheritance: no multiple inheritance :(

Inheritance

```
    final
public class MasterStudent extends Student {

    public void gradeBachelorsWorks() {
        //TODO: implement
    }
}

Student student = new MasterStudent();
student.getYearOfEducation();

System.out.println(student instanceof Student);
System.out.println(student instanceof MasterStudent);
System.out.println(student instanceof Object);
System.out.println(new Object() instanceof Student);
```

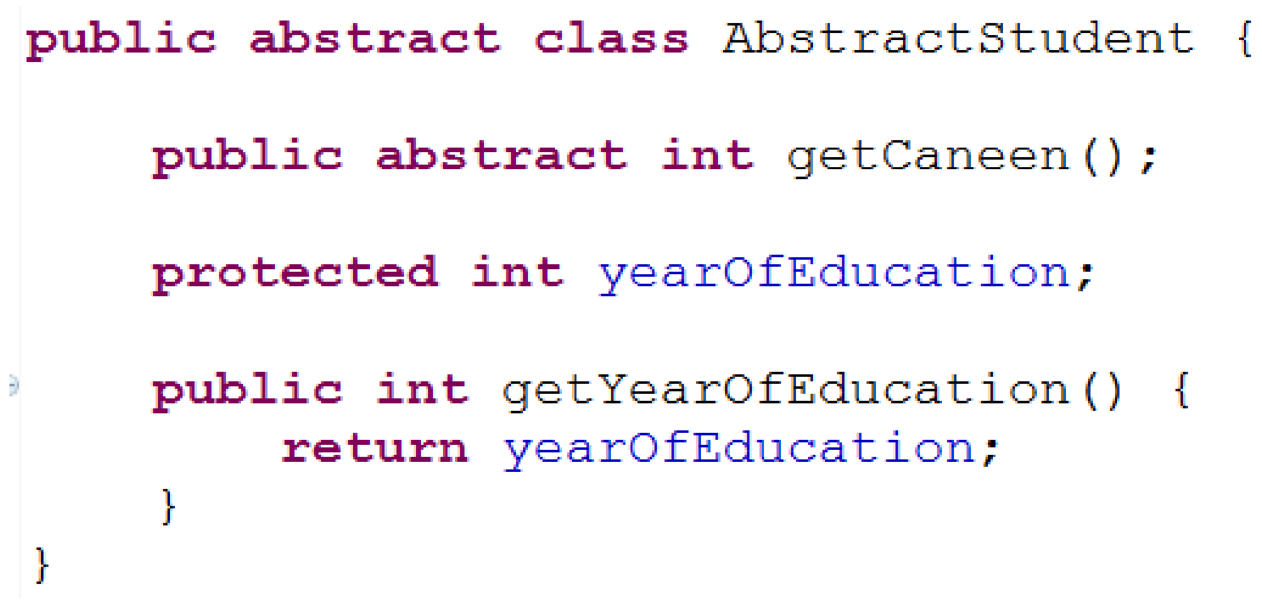
Inheritance: type casting

```java
MasterStudent stu = (MasterStudent)student;
MasterStudent stu2 = (MasterStudent)(new Object());
```

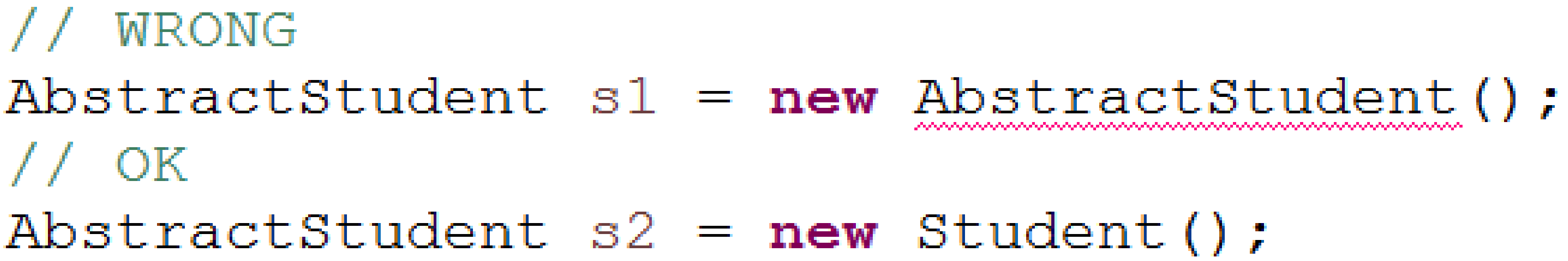
```
Exception in thread "main" java.lang.ClassCastException: java.lang.Object
cannot be cast to MasterStudent
	at Act.main(Act.java:15)
```

```java
if (stu instanceof MasterStudent) {
    MasterStudent mstu = (MasterStudent) stu;
    mstu.gradeBachelorsWorks();
}
```

Inheritance: abstract classes



```java
public abstract class AbstractStudent {

    public abstract int getCaneen();

    protected int yearOfEducation;

    public int getYearOfEducation() {
        return yearOfEducation;
    }
}
```

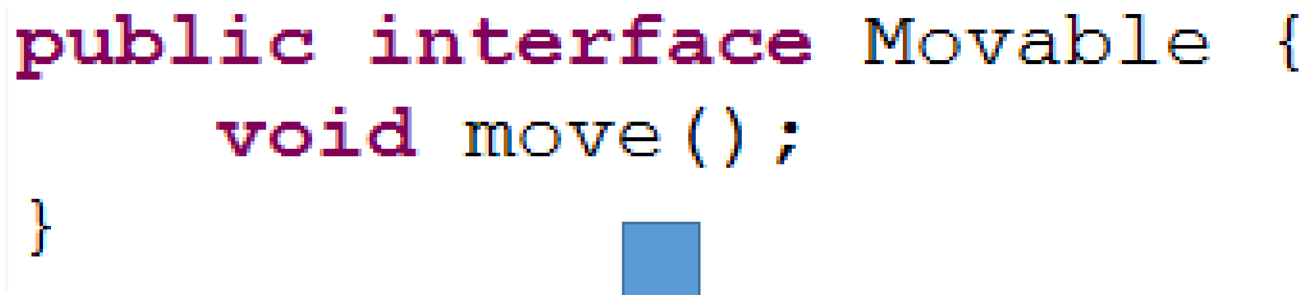


```java
// WRONG
AbstractStudent s1 = new AbstractStudent();
// OK
AbstractStudent s2 = new Student();
```

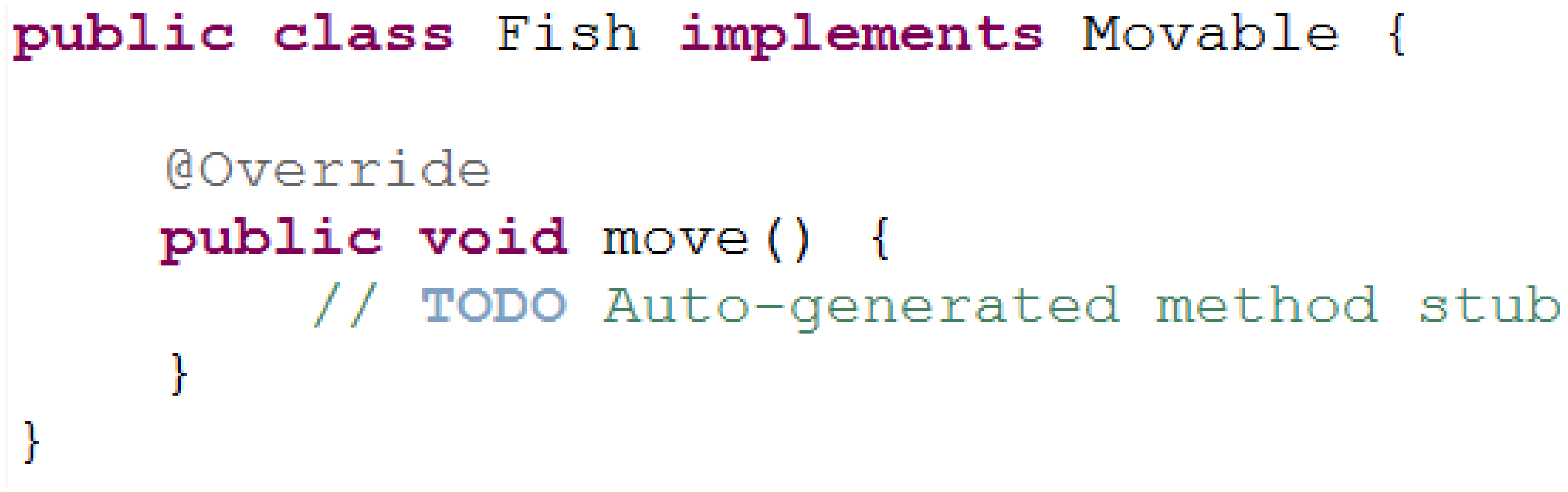
Polymorphism: evolve, or die!

- **Polymorphism (ad hoc)** – is an ability to hide different behaviors under the same interface
    - Abstract classes
    - Interfaces
    - Virtual methods

Polymorphism: interface



```java
public interface Movable {
    void move();
}
```



```java
public class Fish implements Movable {

    @Override
    public void move() {
        // TODO Auto-generated method stub
    }
}
```

Inheritance: virtual methods

```
public class PhdStudent extends MasterStudent {

    private void partyHard() {}

    @Override
    public void gradeBachelorsWorks() {
        super.gradeBachelorsWorks();
        partyHard();
    }

    public final void defendPhD() { }
}
```

Package

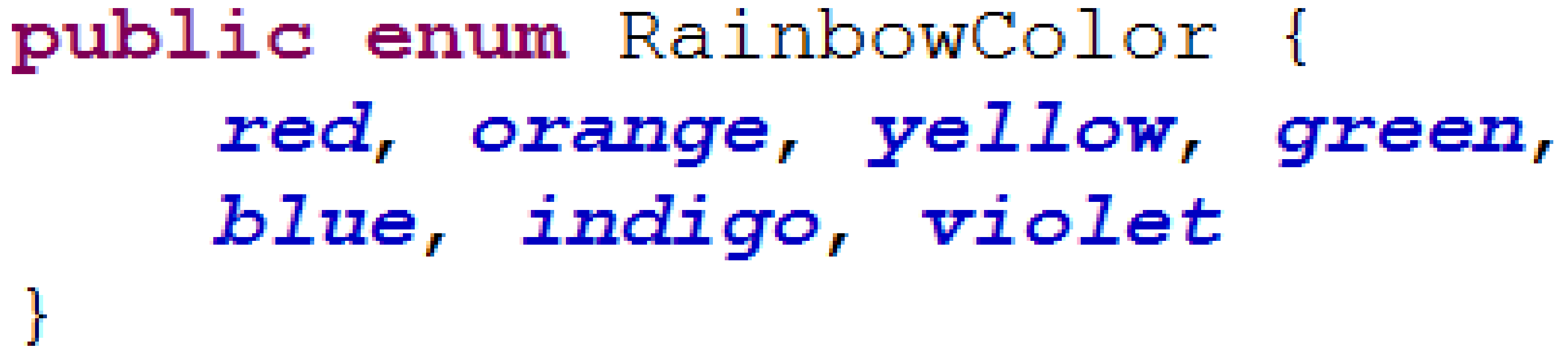
- Package – is a logical structure that helps to organize classes.
- Java packages are mapped to file system.
    - `package ru.innopolis.examples;`
- There are 3 ways to reference a class that is not in your package:

    -
      ```
      java.util.ArrayList list =
                      new java.util.ArrayList();
      ```
    - `import java.util.ArrayList;`
    - `import java.util.*;`

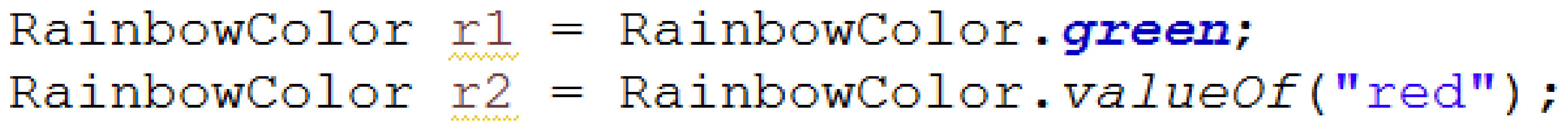
Static – nails for your code

- Sometimes you need **single** class member (field or method) **shared** with all objects (class instances)
- Then you call such a member <u>static</u>
- Entry point (main) is always static

Enum: I know all of them!



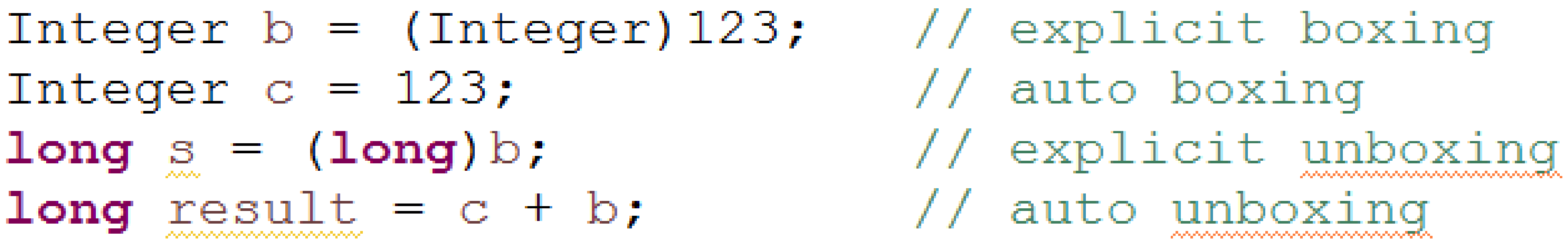
```
public enum RainbowColor {
    red, orange, yellow, green,
    blue, indigo, violet
}
```



```
RainbowColor r1 = RainbowColor.green;
RainbowColor r2 = RainbowColor.valueOf("red");
```

[Auto]boxing and unboxing

- Boxing is a way to connect to worlds of primitive and reference types



```
Integer b = (Integer)123;    // explicit boxing
Integer c = 123;             // auto boxing
long s = (long)b;            // explicit unboxing
long result = c + b;         // auto unboxing
```

INNOPOLIS UNIVERSITY

s.protasov@innopolis.ru