OOP

Tirgul 12

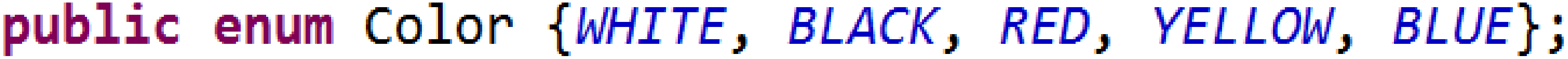
What will we be seeing today?

- Enums
- Ex6

Enums

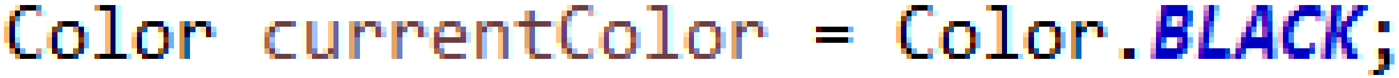
Enums Motivation

- An *enum* type is a special data , used to define a set of predefined constants.
- A variable of an enum type can only be assigned with a constant from the set defined for that enum type.



```
public enum Color {WHITE, BLACK, RED, YELLOW, BLUE};
```

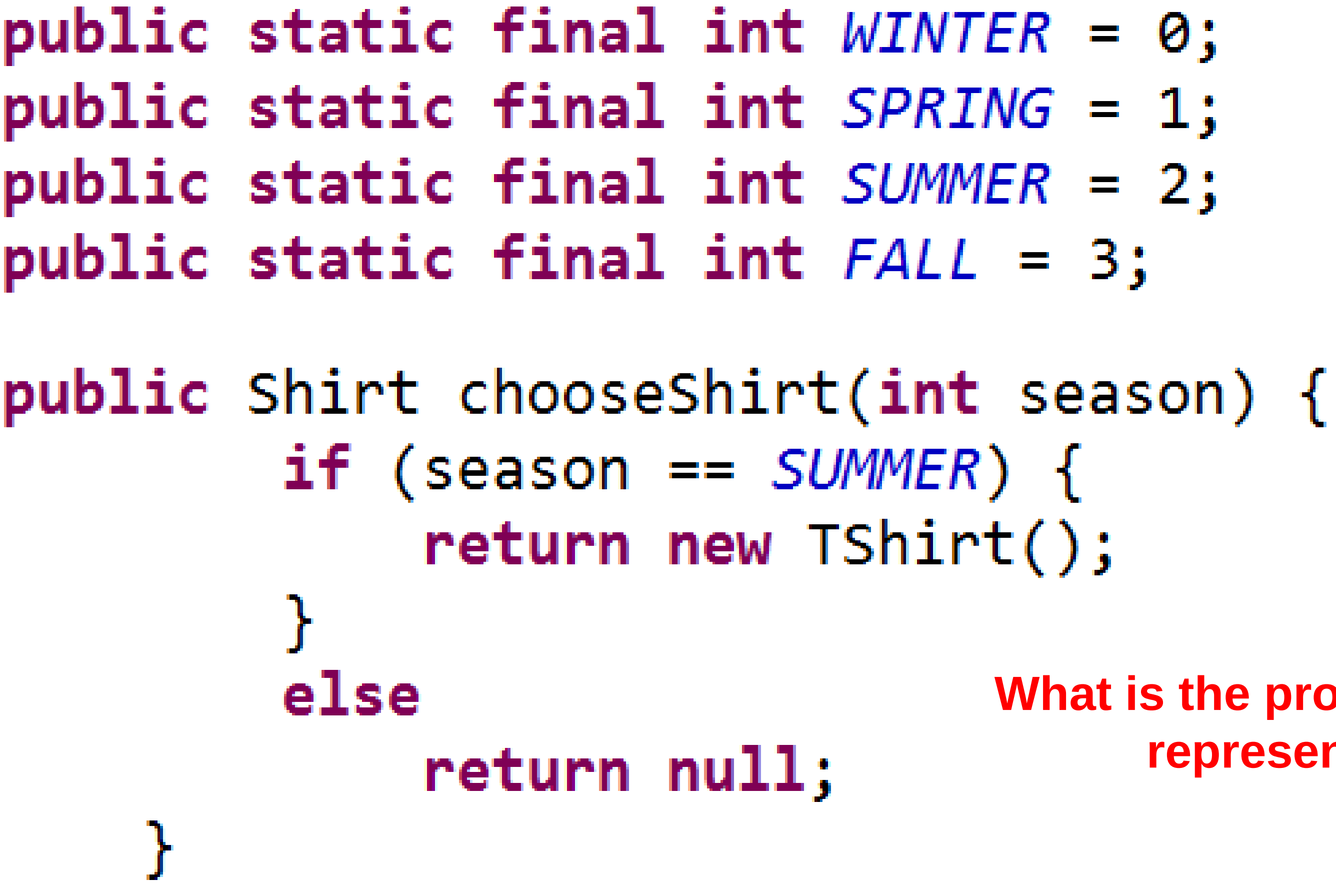
Naming convention



```
Color currentColor = Color.BLACK;
```

Enums Motivation

- Say you want to write a program that requires a representation of the seasons of the year:



```
public static final int WINTER = 0;
public static final int SPRING = 1;
public static final int SUMMER = 2;
public static final int FALL = 3;

public Shirt chooseShirt(int season) {
        if (season == SUMMER) {
            return new TShirt();
        }
        else
            return null;
    }
```

What is the problem with this representation?

Enums Motivation

- **Not typesafe** – season is just an int
  - You can pass any **int** value to chooseShirt()
  - What is WINTER + 3?

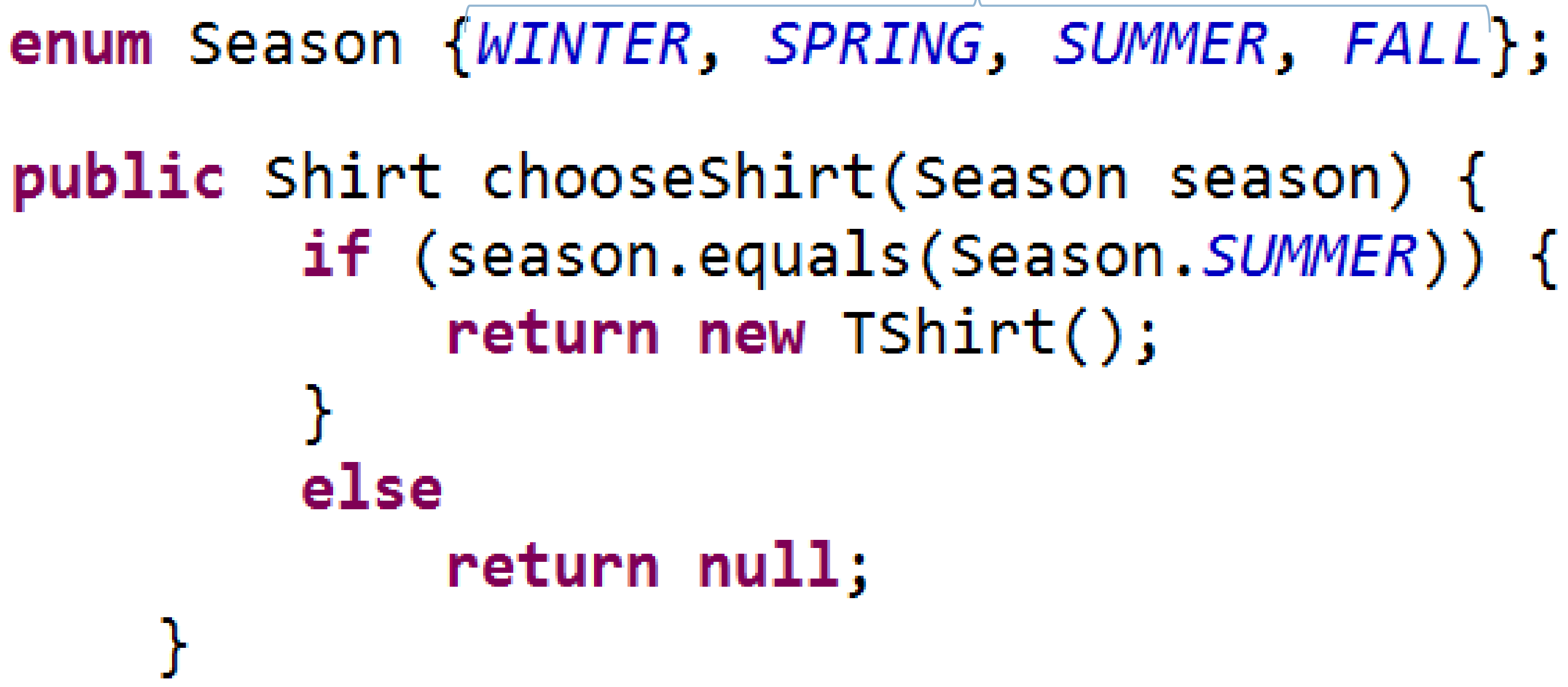
- **Hard to update** – what if you want to add a new season between spring and summer (say, END_OF_SPRING)?
  - Need to change values of other members

- **Uninformative printing**
  - System.*out*.println(SUMMER) prints 2

Enums Solution

- Use Enums!

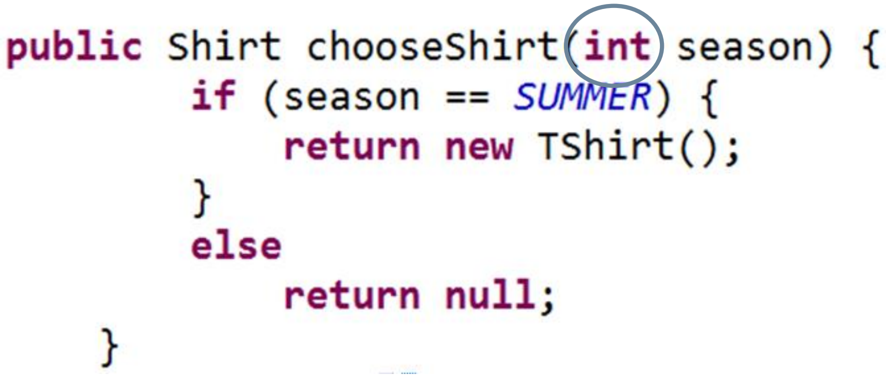
final



```
enum Season {WINTER, SPRING, SUMMER, FALL};

public Shirt chooseShirt(Season season) {
        if (season.equals(Season.SUMMER)) {
            return new TShirt();
        }
        else
            return null;
    }
```

Without Enums Not typesafe

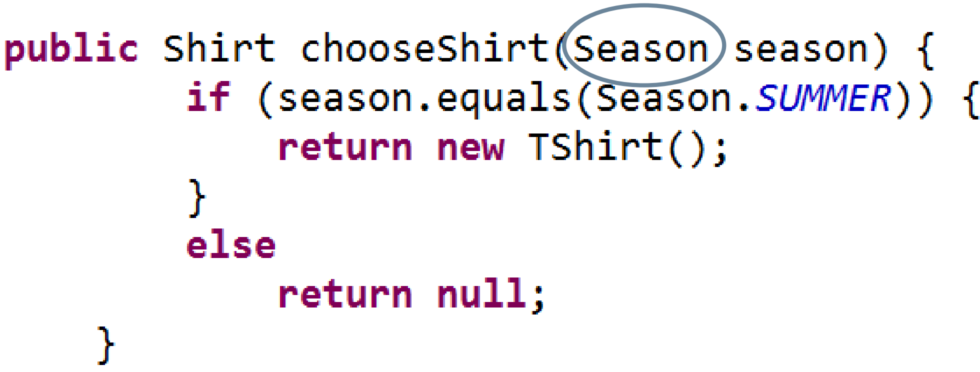
Without
Enums



```
public Shirt chooseShirt(int season) {
        if (season == SUMMER) {
            return new TShirt();
        }
        else
            return null;
    }
```

chooseShirt(1223);

With Enums



```
public Shirt chooseShirt(Season season) {
        if (season.equals(Season.SUMMER)) {
            return new TShirt();
        }
        else
            return null;
    }
```

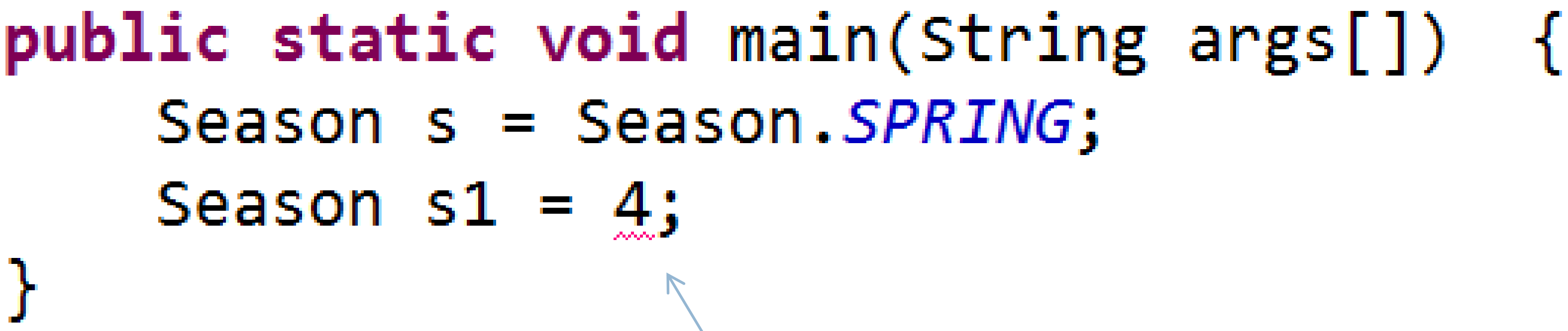
Compilation
Error

chooseShirt(122);

chooseShirt(“SUMMER”);

chooseShirt(Season.SUMMER);

Without Enums Not typesafe



```
public static void main(String args[])  {
    Season s = Season.SPRING;
    Season s1 = 4;
}
```

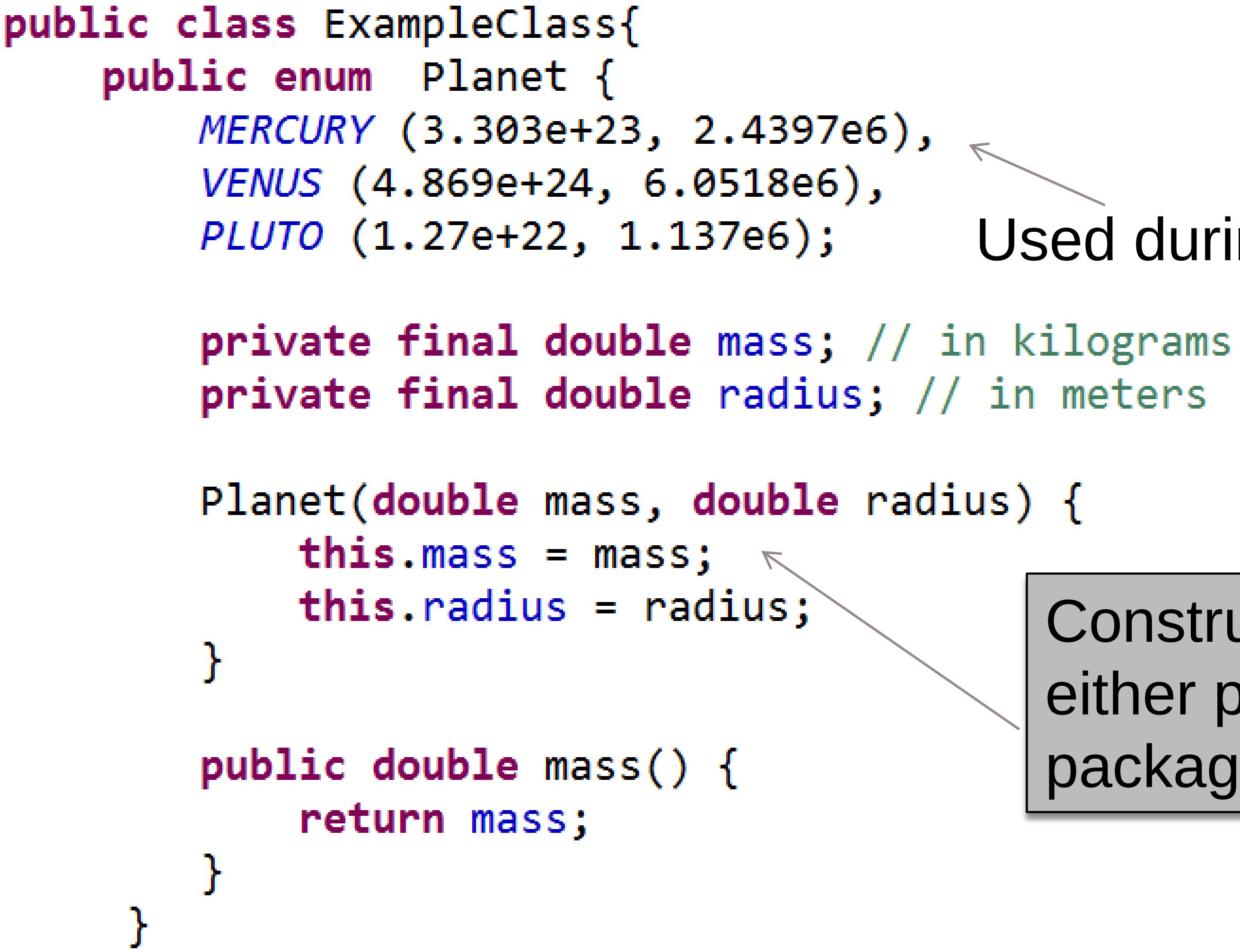
Compilation
Error

Enums Properties

- Enums are actually **java classes**
  - Denoted *enum type (*implicitly extends *java.lang.enum)*
  - *Season.values()* – Iterate over enum values
  - Can have members and methods
  - Constructor must have either **package** or **private** access:
    - Declaring an enum constructor as **public** or **protected** will produce a compile-time error.
    - The default constructor access modifier is **private**
  - You cannot invoke an enum constructor yourself.
  - Enum has its own namespace (Color.BLUE, Season.SUMMER,…).

- Cannot add/change values in runtime
- ≥ Java 5

Enums Complex Example

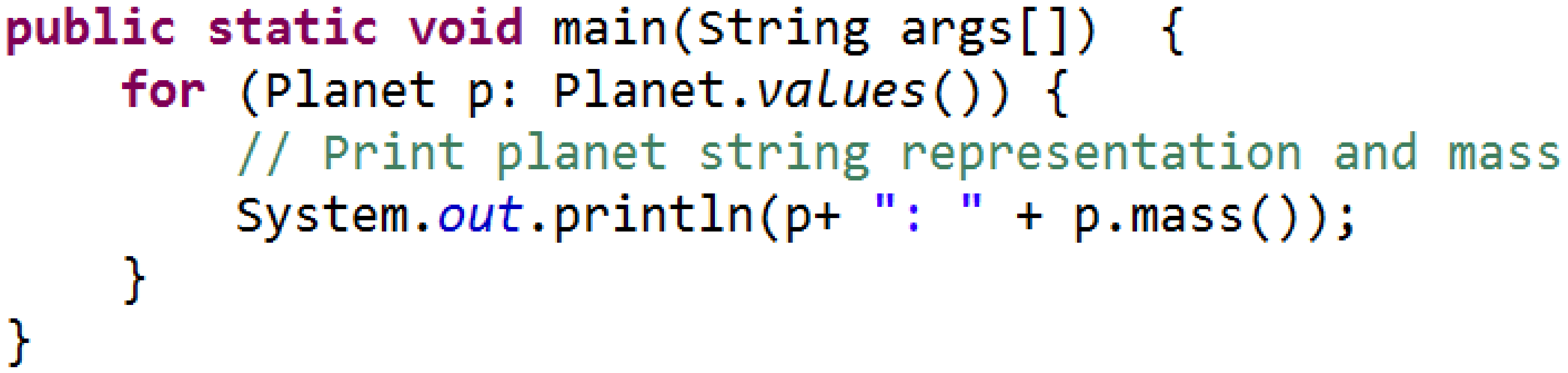


```java
public class ExampleClass{
    public enum  Planet {
        MERCURY (3.303e+23, 2.4397e6),
        VENUS (4.869e+24, 6.0518e6),
        PLUTO (1.27e+22, 1.137e6);

        private final double mass; // in kilograms
        private final double radius; // in meters

        Planet(double mass, double radius) {
            this.mass = mass;
            this.radius = radius;
        }

        public double mass() {
            return mass;
        }
   }
```

Used during construction

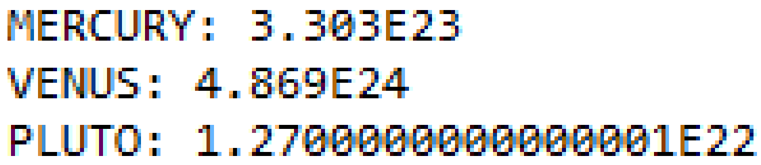
Constructor (declared either private or package)

Enums Complex Example



```
public static void main(String args[])  {
    for (Planet p: Planet.values()) {
        // Print planet string representation and mass
        System.out.println(p+ ": " + p.mass());
    }
}
```

Output:



```
MERCURY: 3.303E23
VENUS: 4.869E24
PLUTO: 1.2700000000000001E22
```

Enums example II

```
public enum Operation {
    PLUS, MINUS, TIMES, DIVIDE;

    // Do arithmetic op’ represented by this constant
    double eval(double x, double y){
        switch(this) {
            case PLUS:   return x + y;
            case MINUS:  return x - y;
            case TIMES:  return x * y;
            case DIVIDE: return x / y;
        }
        throw new UnsupportedOperationException("Unknown op: "+this);
    }
}
```

If we add a new option and not add it in eval(), we might get an exception

String to Enum

```
public class Calculator {

    public static void main(String[] args) {

        String input = args[0];
        Operation o = Operation.valueOf(input);

        System.out.println(o.eval(Integer.parseInt(args[1]),
        Integer.parseInt(args[2])));

    }

}
```

args[0]	args[1]	args[2]
Operation	Number 1	Number 2

Enums example II

We can define an abstract method which has to be implemented by each type

```
public enum Operation {
  PLUS   { double eval(double x, double y) { return x + y; } },
  MINUS  { double eval(double x, double y) { return x - y; } },
  TIMES  { double eval(double x, double y) { return x * y; } },
  DIVIDE { double eval(double x, double y) { return x / y; } };

  // Do arithmetic op represented by this constant
  abstract double eval(double x, double y);
}
```

Enums When to use?

- If you need a fixed set of constants
  - The planets, days of the week, seasons…
  - Other sets where you know all possible values at compile time
    - Choices on a menu, command line flags, etc’
- Pros:
  - Type safety
  - Understandability
  - valueOf method makes it easy to convert from strings to enum values

Enums When not to use?

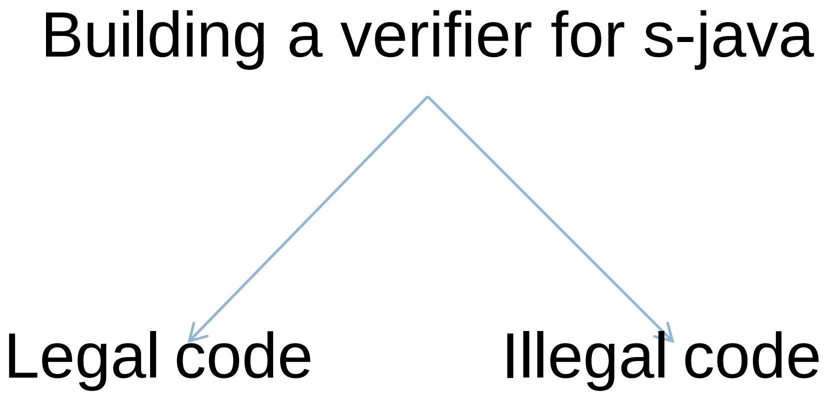
- If your objects are not known in advance (at compile time)
- An Enum defines an object-pool.
  - Each object is instantiated when it is first used.
- Therefore Enums should never be used as value objects or have attributes that get set during usage

ordinal()

- Returns the ordinal of this enumeration constant (its position in its enum declaration)
  - the first constant is assigned an ordinal of zero
- Most programmers will have no use for this method.
- It is designed for use by sophisticated enum-based data structures (where the keys are Enums):
  - EnumSet
  - EnumMap

Ex6

Ex 6



Building a verifier for s-java

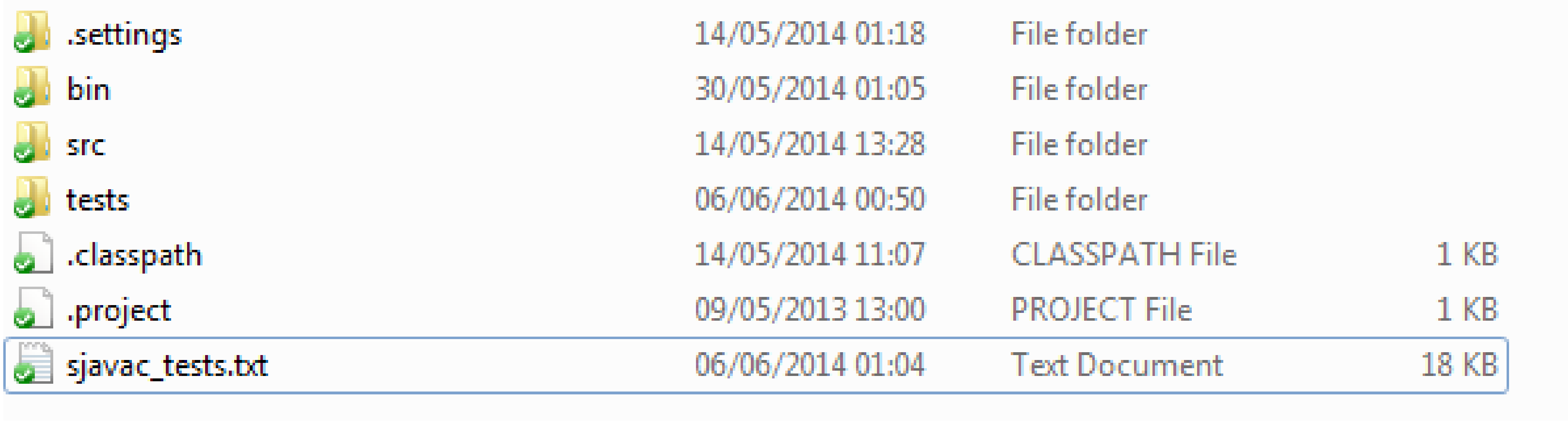
Legal code

Illegal code

System.out.println(0)

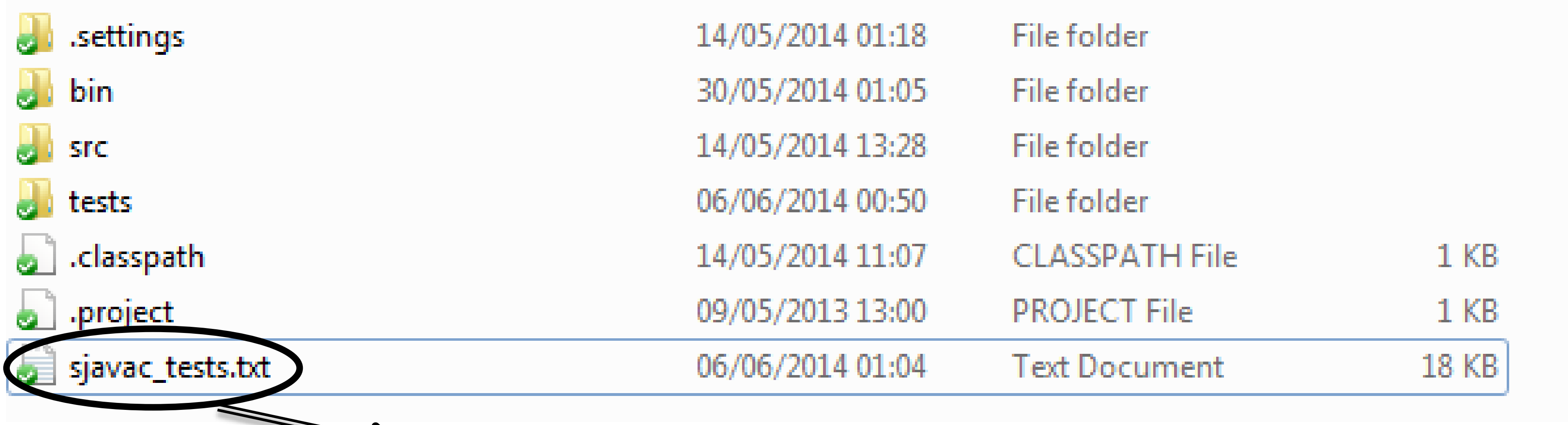
System.out.println(1)

Ex6 – running the testers



.settings	14/05/2014 01:18	File folder	
bin	30/05/2014 01:05	File folder	
src	14/05/2014 13:28	File folder	
tests	06/06/2014 00:50	File folder	
.classpath	14/05/2014 11:07	CLASSPATH File	1 KB
.project	09/05/2013 13:00	PROJECT File	1 KB
sjavac_tests.txt	06/06/2014 01:04	Text Document	18 KB

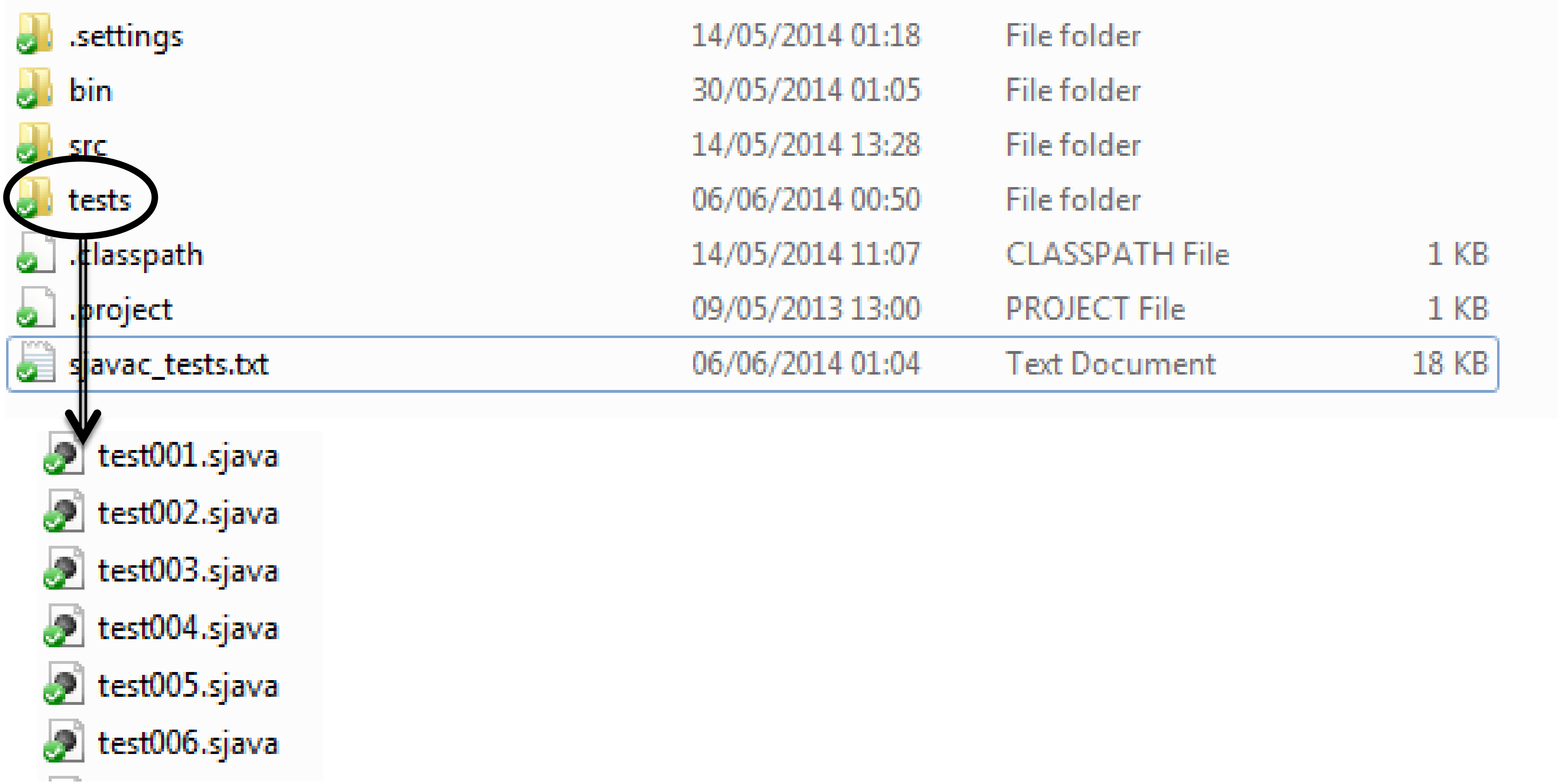
Ex6 – running the testers



.settings	14/05/2014 01:18	File folder	
bin	30/05/2014 01:05	File folder	
src	14/05/2014 13:28	File folder	
tests	06/06/2014 00:50	File folder	
.classpath	14/05/2014 11:07	CLASSPATH File	1 KB
.project	09/05/2013 13:00	PROJECT File	1 KB
sjavac_tests.txt	06/06/2014 01:04	Text Document	18 KB

```
test001.sjava 0 int member test no value
test002.sjava 0 int member test with positive value
test003.sjava 0 int member test with negative value
test004.sjava 0 int member test with zero value
test005.sjava 1 int member test with illegal value: double
test006.sjava 1 int member test with illegal value: string1
test007.sjava 1 int member test with illegal value: string2
test008.sjava 1 final int member test no value
test009.sjava 1 final int member test with positive value

test011.sjava 0 boolean member test no value
```

Ex6 – running the testers



.settings	14/05/2014 01:18	File folder	
bin	30/05/2014 01:05	File folder	
src	14/05/2014 13:28	File folder	
tests	06/06/2014 00:50	File folder	
.classpath	14/05/2014 11:07	CLASSPATH File	1 KB
.project	09/05/2013 13:00	PROJECT File	1 KB
sjavac_tests.txt	06/06/2014 01:04	Text Document	18 KB

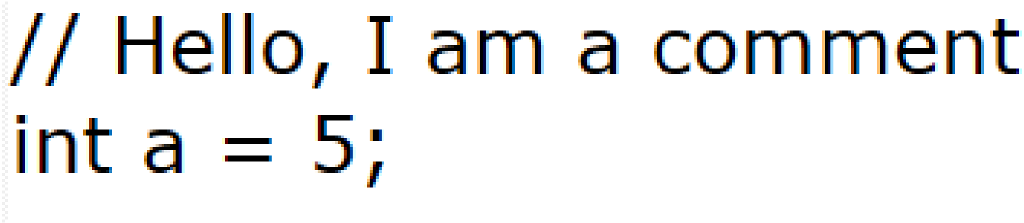
- test001.sjava
- test002.sjava
- test003.sjava
- test004.sjava
- test005.sjava
- test006.sjava

Ex6

- S-java supports:
  - Comments (of a single line)
  - Variables
  - Methods

Ex6 - Comments

- One line comment



```
// Hello, I am a comment
int a = 5;
```

Ex6 - Variables

Defining primitive variables:

```
type name = value;
```

Ex6 - Variables

Defining primitive variables:

type name = value;

int
double
boolean
String
char

Ex6 - Variables

Defining primitive variables:

```
type name = value;
```

```
foo
_foo
foo90
```

Ex6 - Variables

Defining primitive variables:

type name = value;

agrees with the type

Ex6 - Variables

Defining primitive variables:

type name = value;

agrees with the type

Can be:

1) a number
2) Existing initialized variabl

Ex6 - Variables

Defining primitive variables:

type name = value;

agrees with the type

Can be:

int a = 9;

1) a number
2) Existing initialized variable

Ex6 - Variables

Defining primitive variables:

type name = value;

agrees with the type

Can be:

1) a number
2) Existing initialized variable

```
int a = 9;
int b = a;
```

Ex6 - Methods

May contain:

1) Local variable declaration s

2) Local and members variable assignments

3) Call to another function

4) If\while blocks

5) Return statements

Ex6 – Thoughts about the design

- □ **Parser** – can tell whether line is legal (for example, no “;”), and identify the type of the line:
  - New member/local variable? Call to a method?
  - Method definition? Starting a block? …

  Check the validity
- □ **Expressions** – define a method, call to a method, define a variable…
- □ **Variables** – each variable has a different regular expression.

Where do we check if\while blocks?

Ex6 – should it compile or not?

- Run school solution!

  ```
  ~/bin/ex6school file.sjava
  ```

- Think java
- Ask in the forum