PGR112 – Step 3: ArrayList, Scanner, and debugger

Object oriented programming

Bogdan Marculescu / bogdan.marculescu@kristiania.no

Agenda

- ArrayList
- Imports
- Scanner
- User-input
- Debugger
- Wrapper-classes

Status

- Step 1: Development Environment. Hello World.
- Step 2: Methods (loops, if, array, String etc)
- Step 3: Classes and Objects. Encapsulation.
- This step: ArrayList, Scanner, user-input, debugger
- Step 5: Reading from a File. Exception handling
- Step 6: Large Task(with optional tasks)

…

How is it going so far?

- Too difficult?
- Too easy?
- Just right?

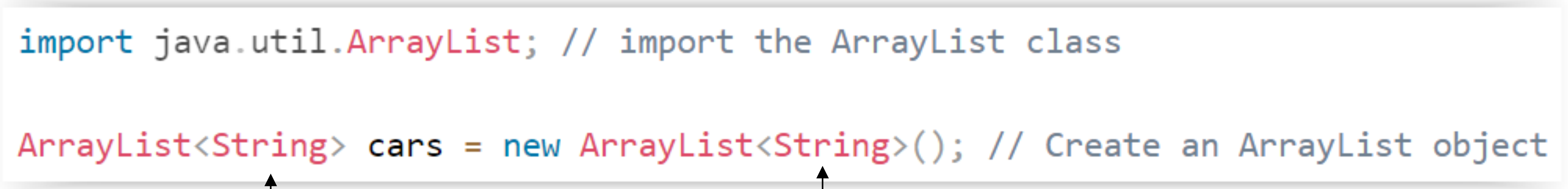
ArrayList

- In the extra tasks in step 3, we put books in a register. The registry used an array to hold the books.
- The disadvantage of arrays is that they have a pre-defined size□
- We do not have this limitation with ArrayList. We can use it to
  - Insert objects
  - Collect
  - Go through
  - Remove

- In an easier way than with the use of an array!

Create

- Hmm, how do we create an ArrayList, I wonder?
- Yes, of course with new. We are going to create an object!
- But let's take a closer look at the syntax (example from w3schools):



```java
import java.util.ArrayList; // import the ArrayList class

ArrayList<String> cars = new ArrayList<String>(); // Create an ArrayList object
```

State the type

Not Necessary

Import

- We import classes we need via import statements at the top of our class.
- In this case, we need to import ArrayList.
- When we import, we specify in which package the class is located. Classes must therefore have unique names within a package.
- If we want to import all classes in one package, then we can e.g. do as follows:
- Import java.util. *;
- But it is considered unadvisable (hides what we really need).

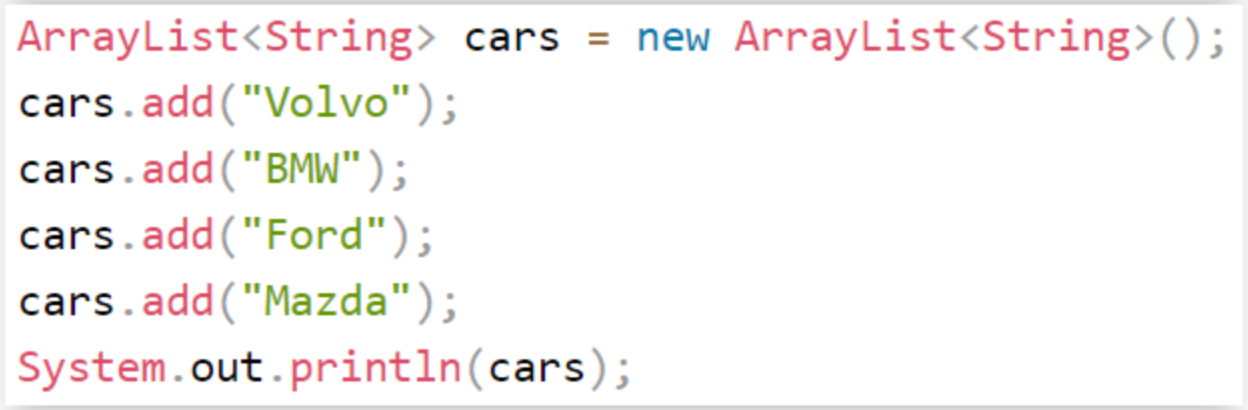
- You can read more about packages here.

Import with IntelliJ

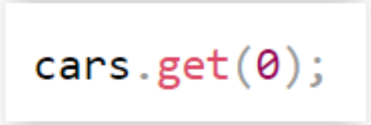
- IntelliJ can help us when we need to import something.
- It's easier than manually entering imports yourself.
- Will be demonstrated in the next demo by ArrayList.

Back to ArrayList

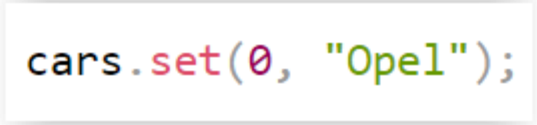
- Add element, retreive, change, etc. (Same link)



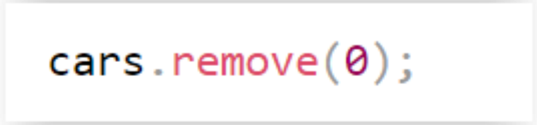
```java
ArrayList<String> cars = new ArrayList<String>();
cars.add("Volvo");
cars.add("BMW");
cars.add("Ford");
cars.add("Mazda");
System.out.println(cars);
```



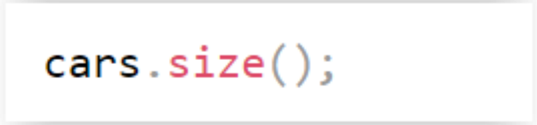
```java
cars.get(0);
```



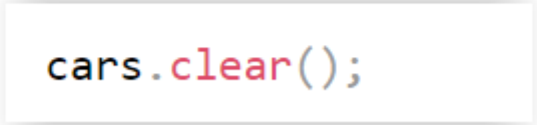
```java
cars.set(0, "Opel");
```



```java
cars.remove(0);
```



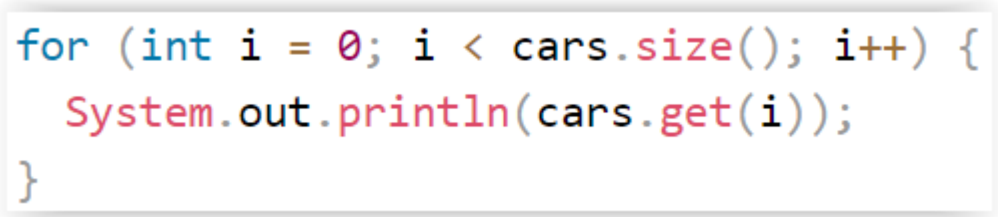
```java
cars.size();
```



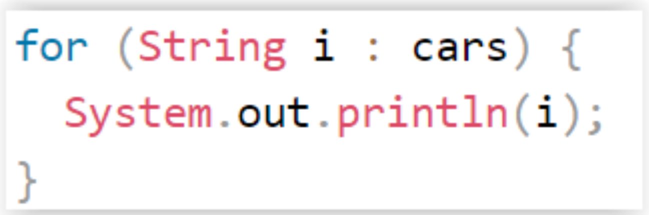
```java
cars.clear();
```

Going through the elements

- Works very similarly as we know from array. You can use for or for-each:



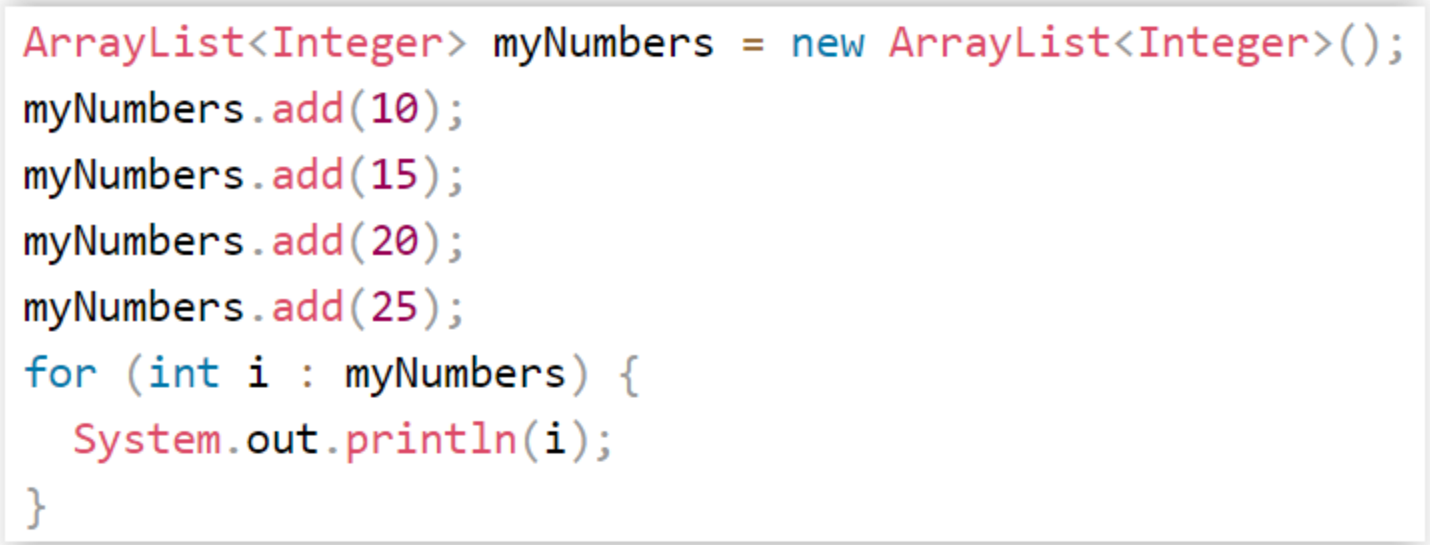
```
for (int i = 0; i < cars.size(); i++) {
  System.out.println(cars.get(i));
}
```



```
for (String i : cars) {
  System.out.println(i);
}
```

ArrayList and objects

- An ArrayList holds objects and not primitive data types. But what if we want to take care of, for example, someone int?
- Fortunately, we have something called wrapper classes for the primitive data types:
  - Integer
  - Character
  - Boolean
  - Double
- …
- There are classes that are made to be able
- keep in a primitive data type and offer
- functionality (methods) related to the value
- (return the value, convert to String etc).

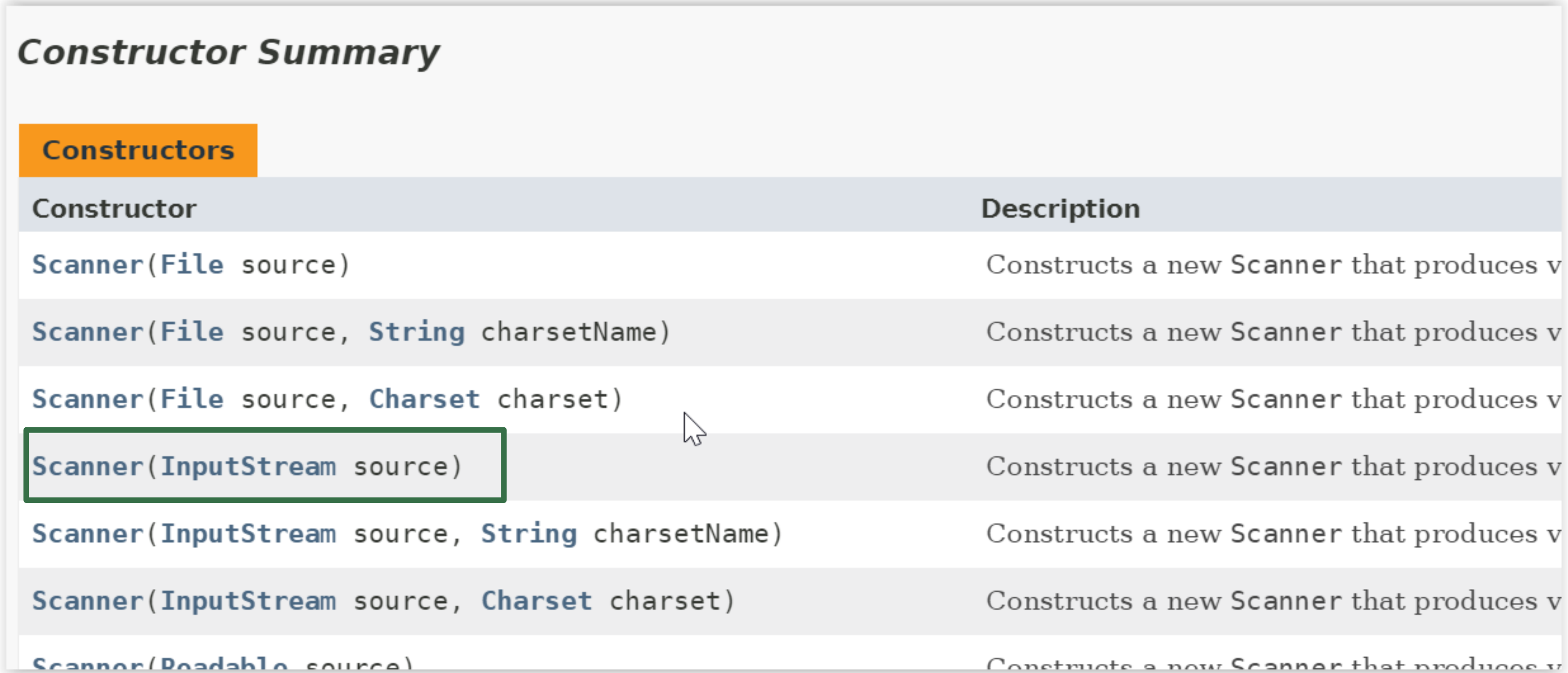


```
ArrayList<Integer> myNumbers = new ArrayList<Integer>();
myNumbers.add(10);
myNumbers.add(15);
myNumbers.add(20);
myNumbers.add(25);
for (int i : myNumbers) {
  System.out.println(i);
}
```

Demo - ArrayList

Scanner

- So, what if we need input from a user?
- One way is already provided by Java: Scanner.
- We are going to create an object of class Scanner. Which constructor to choose?



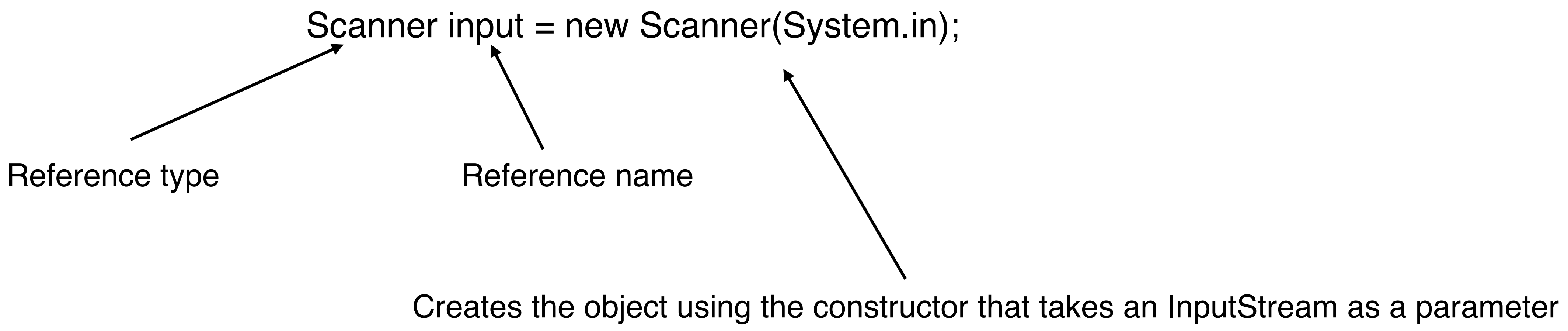
Constructor Summary

Constructors

Constructor	Description
Scanner(File source)	Constructs a new Scanner that produces v
Scanner(File source, String charsetName)	Constructs a new Scanner that produces v
Scanner(File source, Charset charset)	Constructs a new Scanner that produces v
Scanner(InputStream source)	Constructs a new Scanner that produces v
Scanner(InputStream source, String charsetName)	Constructs a new Scanner that produces v
Scanner(InputStream source, Charset charset)	Constructs a new Scanner that produces v
Scanner(Readable source)	Constructs a new Scanner that produces v

Creating Scanner object that can read from console

- System.in is an InputStream we can use. You can read more about it here.
- Creating a Scanner:

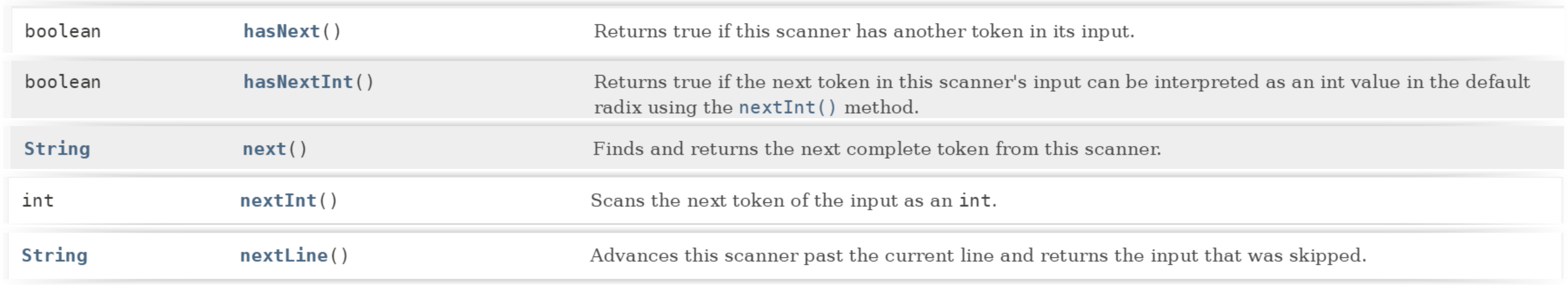


```
Scanner input = new Scanner(System.in);
```

Reference type
Reference name
Creates the object using the constructor that takes an InputStream as a parameter

So how does Scanner (with System.in) work?

- The scanner receives text we type into the console. It divides this input into tokens which by default are split based on white space (space, enter, tab, etc.).
- We can go through tokens using various next methods. In our case, the scanner will wait for input if it is empty and we call next (). The program thus stops waiting for input.
- We also have methods to check if there are more tokens left, and if they are of the type we expect.



`boolean`	`hasNext()`	Returns true if this scanner has another token in its input.
`boolean`	`hasNextInt()`	Returns true if the next token in this scanner's input can be interpreted as an int value in the default radix using the `nextInt()` method.
`String`	`next()`	Finds and returns the next complete token from this scanner.
`int`	`nextInt()`	Scans the next token of the input as an `int`.
`String`	`nextLine()`	Advances this scanner past the current line and returns the input that was skipped.

Demo - Scanner

Problems

- For a Java application to run, it must compile. The compiler makes sure that the person who wrote the program has followed all the rules that must be followed.
- The compiler will report all possible compilation errors, and IntelliJ shows us where the errors are in the code.
- When a program has a main method, and the compiler is satisfied, then we can run the program.
- However, we may still have errors while running our program. Our program may behave differently than intended.
- How to find out what is wrong?

Debug using printing to console

- We like to have an assumption about which area the error is in. We can enter debug printing to check if we are actually running the code lines we expect, and we can print values on relevant variables.
- Demo.

Debugger

- But we can use another method: The debugger.
- Every IDE, with self-respect, has a debugger. With the debugger we can step through our code, step by step.
- We mark one (or more) code lines we want to stop at (break point).
- For each code line we can examine values of both local variables and fields.
- Demo.
- And you can read here (includes video, if you prefer)

Avslutning

- The goals of this step were:
  - I can use Scanner to receive user input.
  - I can use an ArrayList to
    - Handle objects
    - go through objects
  - I can create a user menu using loops
  - I have become a little familiar with the debugger in IntelliJ

- Good luck with the tasks!

- Remember that you have help available all week