Introduction to Java
Part 3

LASSONDE SCHOOL OF ENGINEERING | YORK UNIVERSITÉ UNIVERSITY | EECS ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

Creating Objects

1. Create a new object using the keyword “new”
2. Along with “new”, call the constructor to initialize object attributes
3. Typically, assign the object to a declared reference variable

ClassName identifier = new *ClassName*(*args*);

- E.g.: Blink b = new Blink(true);

String

- Sequence of characters
- Non-primitive (i.e., object) data type
- Read-only objects (recreated but not modified)
  - Any “changes” are actually new objects initialized with the new value
- Strings can be initialized like objects or primitives:

```
String name = new String(“My name is Steven”);
String name = “My name is Steven”;
```

  - The compiler replaces the “short form” with the proper (i.e., object) initialization statement

Concatenation and Indexing

- Strings can be concatenated (i.e., joined) using "+" operator:

```
String s = "EECS" + "1021";
```

- String indexes are numbered 0 to length-1

```
String: EECS1021
Index:  01234567
```

BankAccountV2

- Added withdrawal amount check
- Added toString method
- Updated main method
- Code demonstrated in lecture

Useful String Methods

- length(): the number of characters in this String
- charAt(index): the char at the passed index
- substring(start, end): a new String containing only the characters at the index from start (inclusive) to end (exclusive)
- trim(): a new String with the same characters, but without leading and trailing whitespace
- equals(*otherString*): true iff the current String and otherString are identical
- indexOf(*otherString*): the index of the first occurrence of *otherString* in this String object
- split(*delimiter*): an array of all Strings in this String that were separated by *delimiter*

Parsing Strings as Numbers

- Each primitive type has its own wrapper class
- Wrapper classes store (“wrap”) its associated value in an object and contain static methods
- For example:
  - double n = Double.parseDouble(“8.3”); // n = 8.3

Console Output

- System.out.print(“hello”);
  System.out.print(“bye”);
  - Outputs:
    hellobye

- System.out.println(“hello”);
  System.out.println(“bye”);
  - Outputs:
    hello
    bye

Formatted Output

- Use System.out.printf("*format string*", *args*)
- Similar to sprintf in Matlab
- Format string represents the output, but with placeholders and formatting options for the passed arguments (see Formatter API)
- Example:
  - System.out.printf("PI to 2 decimal places is %.2f.", Math.PI);
    - Outputs: PI to 2 decimal places is 3.14.

File Output

- Use the PrintStream class to create an object representing an output file
  - The file will be created if it doesn't already exist
  - The file will be overwritten if it already exists, so be careful when doing this
- Use the print, println, printf methods as before
- Example:

```
PrintStream output = new PrintStream(new File("output.txt"));
output.println("PI to 2 decimal places is %.2f.", Math.PI);
output.close();
```

  - This will create a file called "output.txt" with the contents "PI to 2 decimal places is 3.14."

Console Input

- Use the Scanner class
- Scanner has methods to read in the next…
  - int, double, char, etc. (i.e., primitive value)
  - String or an entire line of text as a String object
- Also has methods to test if there is more input to read-in
- Example:

```
Scanner input = new Scanner(System.in);  // from console
int value = input.nextInt();  // user input
input.close();
```

File Input

- Similar to console input, but you pass a File object as an argument

- Example:

```
Scanner fileInput = new Scanner(new File("input.txt"));
String firstLine = fileInput.nextLine(); // first line from input.txt
fileInput.close();
```

- Potential errors for file I/O:
  - Missing file, file not readable, etc.
  - Must add "throws" clause to method declaration
  - More on handling these exceptions in two weeks

PowerCalculatorV2

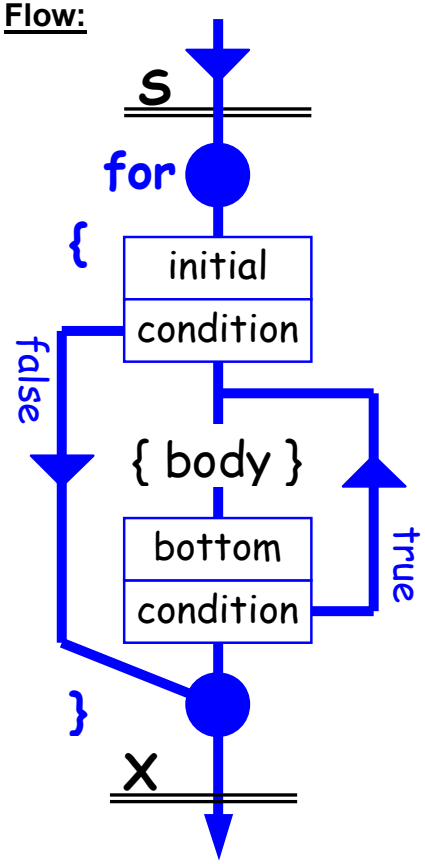
- Added console input and file output
- Code demonstrated in lecture

For-Loops

- Loop body
  - Statements to be executed iteratively (i.e., to be looped)
- Initialization statement (optional)
  - Executed once, when the loop is first encountered
  - Used to declare and/or initialize any variables used within the loop body (be careful of variable scope)
- Boolean condition to continue iteration (i.e., looping)
  - Similar to the if-statement condition
  - Loop body is executed if the condition holds (i.e., is true)
- Update statement (optional)
  - Update variables/state at the end of each iteration (i.e., loop)

For-Loops

Flow:



Syntax:

```
Statement-S
for (initial; condition; bottom)
{
   body;
}
Statement-X
```

Algorithm:

1. Start the `for` scope
2. Execute `initial`
3. If `condition` is `false` go to `9`
4. Start the body scope `{`
5. Execute the body
6. End the body scope `}`
7. Execute `bottom`
8. If `condition` is `true` go to `4`
9. End the `for` scope

For-Loops: Example

- Output the numbers from 1..100
- Sequential

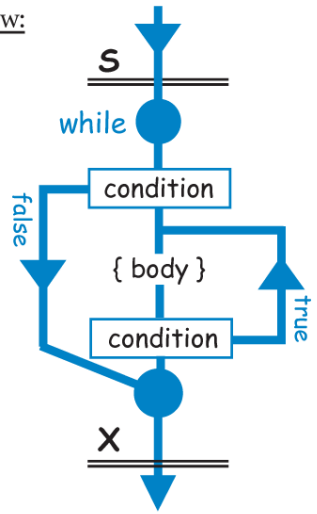
```
System.out.println("1");
System.out.println("2");
System.out.println("3");
…
```

- Iterative

```
for (int count = 1; count <= 100; count++)
{
   System.out.println(count);
}
```

Other Loops

Flow:

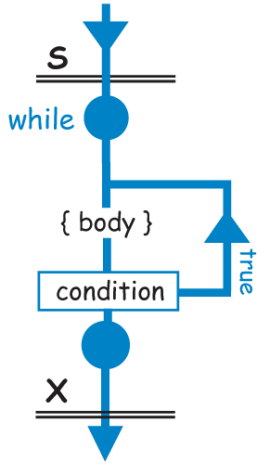


Syntax:

```
Statement-S
while(condition)
{
    body;
}
Statement-X
```

Algorithm:

1. If `condition` is `false` go to 6
2. Start the body scope {
3. Execute the body
4. End the body scope }
5. If `condition` is `true` go to 2
6. End the `while` scope



Syntax:

```
Statement-S
do
{
    body;
}
while (condition)
Statement-X
```

Algorithm:

1. Start the body scope {
2. Execute the body
3. End the body scope }
4. If `condition` is `true` go to 1
5. End the `while` scope

For-Loops with Console Input

- Add non-negative ints, stop when a negative integer is entered:

```
int sum = 0;
Scanner input = new Scanner(System.in);
for(int n = input.nextInt(); n >= 0; n = input.nextInt())
{
        sum += n;
}
input.close();
```

For-Loops with File Input

- Read-in lines from a file and output them to the console

```
Scanner fileInput = new Scanner(new File("input.txt"));
for(String line; fileInput.hasNextLine(); )
{
        line = fileInput.nextLine();
        System.out.println(line);
}
fileInput.close();
```

BankAccountV3

- Added file I/O
- Code demonstrated in lecture