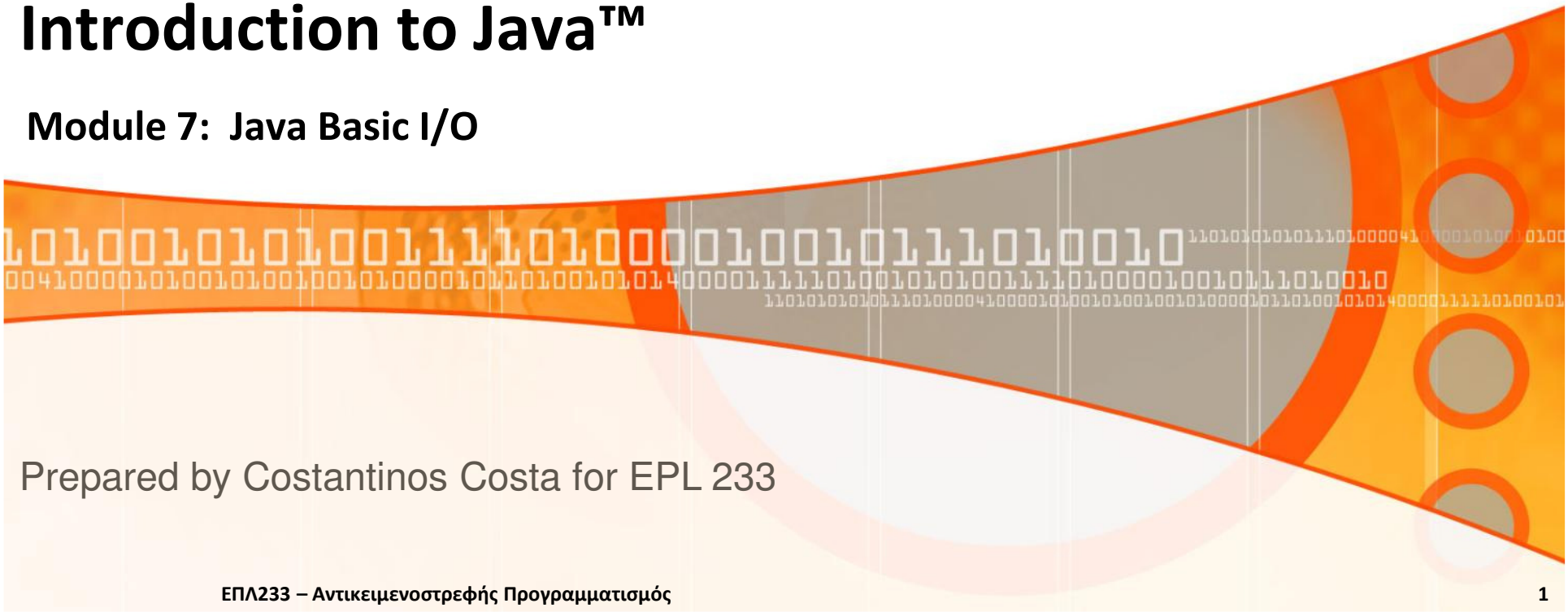
Introduction to Java™

Module 7: Java Basic I/O



Prepared by Costantinos Costa for EPL 233

Basic I/O

- **Command line arguments**
  - Using command line is a useful way to read in few user input values
  - Two significant limitations of using command line arguments
    - Unwieldy to enter large amounts of data using this method
    - Command line parameters must be specified *before* the program starts executing
- **Standard output**
  - Most popular operating systems provide a device called *standard output* that represents an abstract place to send output
  - Typically standard output is associated with the Terminal (Mac OS X), Command Prompt (Windows), or Shell (Unix).
    - We'll use the term *terminal window*
    - We aren't required to associate standard output with the terminal window
    - We can *redirect* standard output to a file for permanent storage
  - The System.out.println method sends its output to standard output

Basic I/O (continued)

- **Standard input**
  - Most popular operating systems also provide a device known as *standard input* that represents an abstract place to receive input
  - Standard input overcomes the 2 main obstacles of command lines
    - Easy to deal with large quantities of data
    - Input data is read in while the program is executing
      - Enables interactive programs which prompt the user for information
  - Standard input is typically associated with the terminal window
    - The user enters data by typing at the keyboard in the terminal windows
    - Java provides support for reading in data from standard input
      - A bit cumbersome to use
      - We provide a custom-made library called StdInput.java

StdInput.java

```
/*************************************************************************
 * Compilation: javac StdInput.java
 * Execution: java StdInput
 *
 * Supports reading variables of type int, double, String, boolean,
 * long, or float from standard input.
 *************************************************************************/
import java.io.IOException;
public class StdInput { private static int c = ' ';
    private static final int EOF = -1;
    // can't create an instance of this class
    private StdInput() { }
    // is the current character whitespace?
    private static boolean isBlank() { return Character.isWhitespace((char) c); }
    // is it at end of the file already?
    private static boolean isEOF() { return c == EOF; }
    // return EOF if end of file or IO error
    private static void readC() { try { c = System.in.read(); } catch(IOException e) { c = EOF; } }
    // is there more input?
    public static boolean isEmpty() { while (!isEOF() && isBlank()) readC(); return isEOF(); }
    // read a token - use StringBuffer for efficiency
    public static String readString() {
            StringBuffer s = new StringBuffer();
            // eat up whitespace
            while (!isEOF() && isBlank()) readC();
            // now get the string
            while (!isEOF() && !isBlank()) { s.append((char) c); readC(); }
            if (s.length() == 0) throw new RuntimeException("Tried to read from empty stdin");
            else return s.toString(); }
    public static int readInt() { return Integer.parseInt(readString()); }
```

StdInput.java (continued)

```
public static double readDouble() { return Double.parseDouble(readString()); }
public static float readFloat() { return Float.parseFloat(readString()); }
public static double readLong() { return Long.parseLong(readString()); }
public static boolean readBoolean() {
      String s = readString().toLowerCase();
      if (s.equals("true") || s.equals("yes") || s.equals("1")) return true;
      if (s.equals("false") || s.equals("no") || s.equals("0")) return false;
      throw new RuntimeException(s + " is not a valid boolean value"); }
// test client
public static void main(String[] args) {
// read in an integer and print it out
System.out.print("Enter an integer: ");
int x = StdInput.readInt();
System.out.println("Your integer was " + x);
System.out.println();
// read in an double and print it out
System.out.print("Enter a double: ");
double y = StdInput.readDouble();
System.out.println("Your double was " + y);
System.out.println();
// read in a String and print it out
System.out.print("Enter a string: ");
String z = StdInput.readString();
System.out.println("Your string was " + z);
System.out.println();
// read in a boolean and print it out
System.out.print("Enter a boolean: ");
boolean b = StdInput.readBoolean();
System.out.println("Your boolean was " + b);
System.out.println(); } }
```

StdInput.java (continued)

- The table below summarizes all of the available operations with the StdInput library

COMMAND	RETURN TYPE	PURPOSE
isEmpty	boolean	test if the input stream is empty
readBoolean	boolean	read in yes, 1, or true for true, no, 0, or false for false
readInt	int	read in an integer
readLong	long	read in a long integer
readDouble	double	read in a double precision real number
readFloat	float	read in a single precision real number

Manipulating Standard Input/Output

- Redirecting standard output
  - java MyProgramOut > MyData.txt
- Redirecting standard input
  - java MyProgramIn < MyData.txt
  - StdInput library usage example

```
while (!StdInput.isEmpty()) {
      double x = StdInput.readDouble();
   n++; sum += x; sum2 += x*x;
}
```

- Linking two programs
  - Through the use of OS pipes
  - java MyProgramOut | java MyProgramIn

Simple Graphics

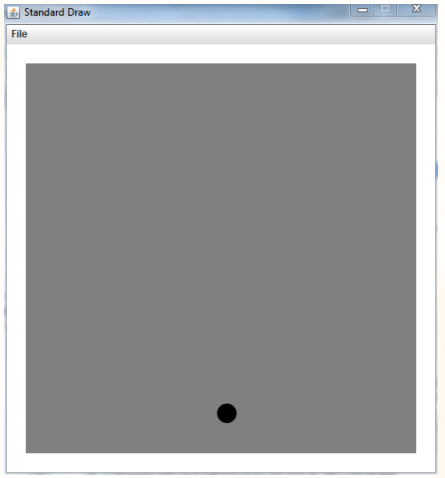
- Standard output is a very useful abstraction to represent a generic text output stream
  - Enables us to output to the terminal window or a file without having to edit or re-compile
  - We want the same type of abstraction for graphics output
    - Produce a picture that we can display on the screen or save to file
    - We also want creating pictures to be as easy as reading with StdInput
- We will use our StdDraw library to produce static and animated graphics
  - It simulates the behavior of a Turtle (like in LOGO) moving in a two dimensional space
  - The Turtle has a pen strapped to its back, drawing while it moves
  - Program StdDraw.java is a version that draws graphics in a window
  - StdDraw.java is a rather complicated program
- In order to use the StdDraw graphics library, put a copy of StdDraw.java in your working directory
  - Include a copy of StdInput.java in your working directory too
  - To see the offered methods study the code of StdDraw

Practice

- In the previous lab you downloaded the Standard Libraries of the Sedgewick and Wayne book.
  - http://www.cs.princeton.edu/introcs/stdlib/
- Write a program that reads from the keyboard and draws simple shapes in the screen
  - Your program should present a menu with the following options:
    - Circle, Square, Triangle
  - Depending on the selection made it should ask the
    - Center (x,y) and radius of the circle
    - The starting point (x,y) and side length of the square
    - The tree points of the triangle
  - Draw them on screen
  - Use the redirection of Standard input and output to create your drawings

Task1

- **Go to** http://www.cs.princeton.edu/introcs/stdlib/
- **Download Average.java** and **BouncingBall.java**
- **Run** the codes and see the results.
- **Average.java:**
  - Reads in a sequence of real numbers, and computes their average.
  - Dependencies: StdIn.java StdOut.java
- **BouncingBall.java:**
  - Implementation of a 2-d bouncing ball in the box from (-1, -1) to (1, 1).
  - Dependencies: StdDraw.java



Task2

- Based on the bouncing ball example, **create new** project called BouncingBalls. At Bouncing balls we should extend the original code in order to have many balls and bounce to each other and the wall.
- **Note:** In this case whenever have collision between two ball the velocity should be reversed
    - e.g velocity=-velocity