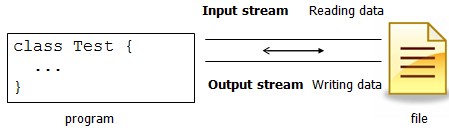
#### ICS 102 Lab 03: Console I/O

Objectives:

* Introduction to I/O Streams
* Review of Console Output
* Console Input using **Scanner**

#### I/O Streams

A *stream* is the flow of data between a program and some I/O device or file.



A stream can represent many different kinds of sources and destinations, including disk files, devices, other programs, and memory arrays.

Streams support many different kinds of data, including simple bytes, primitive data types, and objects.

**The standard I/O streams**

Among the facilities provided by the **java.lang.System** class are the standard input, standard output, and error output streams. The standard input streams are used for reading character data. The standard output streams are used for printing.

* **System.in** is an InputStream object that is typically connected to keyboard input of console programs.
* **System.out** is a PrintStream object. It normally outputs the data you write to it to the computer screen.
* **System.err** is a PrintStream object. It normally outputs the data you write to it to the computer screen. By convention, this output stream is used to display error messages.

The **java.lang.System** class creates these I/O streams automatically for us when our application begins execution. Each of these streams is *public* and *static* so that we can access them directly without having to create an instance of the **System** class.

Together, the monitor and keyboard are often referred to as the **console**.

**2. Console output**

We have covered console output in the previous labs:

|  |  |
| --- | --- |
| Statement Syntax | Examples |
| System.out.print(String); | System.out.print("Dhahran");  System.out.print("Speed = " + speed + " km/hr "); |
| System.out.print(Expression); | System.out.print(12 + 8); |
| System.out.println(String); | System.out.println("Dhahran");  System.out.println("Speed = " + speed + " km/hr "); |
| System.out.println(Expression); | System.out.println(12 + 8); |
| System.out.printf(String); | System.out.printf("Dhahran%n"); |
| System.out.printf(FormatString, ListOfExpressions); | System.out.printf("x = %d, y = %d, x\*y = %d%n", x, y, x \* y); |

By convention, the **System.err** stream is used to display error messages. Example:

System.err.println("Error: The input file does not exist ");

**3. Using Java's Scanner Class to Read Console Input**

The **java.util.Scanner** class provides a wrapper class that encapsulates an input stream. An object of this class provides a number of methods for reading lines and values of various types.

In Java, the variable **System.in** is an InputStream that is connected to the keyboard. **System.in** is a byte stream so you cannot read from it directly if you want to read character strings. Hence, you must wrap a Scanner object around **System.in** to handle string oriented I/O from the keyboard:

**Scanner scannerObject = new Scanner(System.in);**

A Scanner object breaks its input into tokens using a delimiter pattern, which by default is whitespace. The resulting tokens may then be converted into values of different types using the various next methods:

|  |  |
| --- | --- |
| Method | Effect when reading from keyboard |
| int nextInt() | Skips leading delimiters in the input buffer and then consumes and returns the next token. If the next token cannot be converted to the method’s return type or is out of range, **InputMismatchException** is thrown. Trailing delimiters are not removed from the input buffer. |
| long nextLong() |
| float nextFloat() |
| double nextDouble() |
| String next() | Skips leading delimiters in the input buffer, then consumes and returns the next token as a string. Trailing delimiters are not removed from the input buffer. |
| String nextLine() | Reads the rest of the current line and returns it as a string. The line terminator character at the end of the line is consumed but is not appended to the string. |

**Discarding a Scanner**

When you have finished using a Scanner, you should close it using the close method:

**void close()**

This closes the Scanner and allows Java to reclaim the Scanner's memory.

**Note:** Closing the console is not necessary, but it is a good habit. If we were reading a file, which is common with Scanner, closing it would be important.

**Steps for console based user input:**

1. Use the **System.in** InputStream object to create a **Scanner** object. Since the Scanner class is not in the **java.lang** package, import it by adding one of the following statements at the top of your program:

**import java.util.Scanner;** or **import java.util.\*;**

**Scanner keyboard = new Scanner(System.in);**

1. Display a prompt to the user for the desired data.

**System.out.print( "Please input . . . : " );**

1. Use an appropriate **Scanner** object method to read the input from the user:

|  |  |
| --- | --- |
| Input to be read | Sample statement |
| int | int num1 = keyboard.nextInt(); |
| long | long num2 = keyboard.nextLong(); |
| float | float num3 = keyboard.nextFloat(); |
| double | double num4 = keyboard.nextDouble(); |
| word | String str1 = keyboard.next(); |
| line | String str2 = keyboard.nextLine(); |

1. Do the required processing with the input received from the user.
2. Close the input stream:

**keyboard.close();**

Example:

|  |
| --- |
| **import java.util.Scanner; public class ConsoleInput1{  public static void main(String[] args){  int year1, year2;  String name;  Scanner kybrd = new Scanner(System.in);  System.out.println("Please enter your full name: ");  name = kybrd.nextLine();  System.out.println("Please enter your year of birth: ");  year1 = kybrd.nextInt();  System.out.println("Please enter current year: ");  year2 = kybrd.nextInt();  System.out.println("Mr. " + name + " you are " +**  **(year2 - year1) + " years old.\n");  } }** |

**Problem of nextLine( ) with a left-over line terminator**

The Scanner methods nextInt( ), nextLong( ), nextFloat( ), nextDouble( ), and next( ), do not consume trailing delimiters. If one of these methods reads a token **at the end of the current line**, and then a nextLine( ) is issued, the nextLine( ) will consume the new line character at the end of the current line and return an empty string; the next input line will not be read.

Example:

|  |  |
| --- | --- |
| **import java.util.Scanner; public class ConsoleInput3{  public static void main(String[] args){  String word, sentence;  Scanner kybrd = new Scanner(System.in);  System.out.println("Please enter a single word: ");  word = kybrd.next();  System.out.println("Please enter a sentence: ");  sentence = kybrd.nextLine();  System.out.println("The word is " + word);  System.out.println("The sentence is " + sentence);  } }** | |
| User input: | Effect: |
| happy | Please enter a single word:  **happy** Please enter a sentence:  The word is happy The sentence is |

**Solution:** The solution to the problem depicted above is to introduce a statement, before

**sentence = kybrd.nextLine()**, that will remove or skip the line terminator left in the input buffer by **kybrd.next()**. Use a dummy nextLine() statement:

**kybrd.nextLine();**

|  |
| --- |
| import java.util.Scanner; public class ConsoleInput3{  public static void main(String[] args){  String word, sentence;  Scanner kybrd = new Scanner(System.in);  System.out.println("Please enter a single word: ");  word = kybrd.next();  **kybrd.nextLine();** // dummy  System.out.println("Please enter a sentence: ");  sentence = kybrd.nextLine();  System.out.println("The word is " + word);  System.out.println("The sentence is " + sentence);  } } |

**Reading one character at a time from the keyboard using Scanner**

Java's Scanner class does not have a built in method to read from a Scanner character-by-character. To read a character, read a string and then extract the first character of the string:

**Scanner scanner = new Scanner(System.in);   
char character = scanner.next().charAt(0);**

or

**char character = scanner.nextLine().charAt(0);**

To read exactly one character use:

**Scanner scanner = new Scanner(System.in);**

**scanner.useDelimeter("");   
char character = scanner.next().charAt(0);**

Example:

|  |
| --- |
| **import java.util.Scanner;  public class CharInput{  public static void main(String[] args){  Scanner scanner = new Scanner(System.in);  String str1;  char ch;  System.out.println("Enter a string: ");  str1 = scanner.nextLine();  System.out.println("Enter a character: ");  ch = scanner.nextLine().charAt(0);  boolean flag = str1.contains(ch+"");    System.out.println("It is " + flag + " that \"" + str1 +**  **"\" contains " + ch);  } }** |

**Exceptions**

Exceptions are run-time errors that **may** be generated by the Java system when something goes wrong with a program.

An exception is handled by using a **try-catch block**.

The syntax of try-catch block that is used to catch an exception:

**try{ . . .**

**statementThatMayCauseException;**

**. . .**

**} catch(ExceptionType e){**

**Code to handle the exception**

**}**

If an exception occurs, the statements in the catch block that follow the statement that caused the exception are not executed; control is transferred directly to the catch block. If no exception occurs, all statements in the catch block are executed and the statements in the catch block are not executed.

Examples:

**try{**

**// A statement that may cause InputMismatchException**

**. . .**

**} catch(InputMismatchException e){**

**System.out.err(e); System.exit(1);**

**}**

import java.util.Scanner;

import java.util.InputMismatchException;

class Example2 {

public static void main(String args[ ]) {

int num1, num2, sum;

Scanner kb = new Scanner(System.in);

try {

System.out.print("Enter two integers: ");

num1 = kb.nextInt( );

num2 = kb.nextInt( );

sum = num1 + num2;

System.out.println("sum = " + sum);

}

catch (InputMismatchException e)

{

System.out.println("Error: Wrong input");

}

}

}

**Laboratory Tasks:**

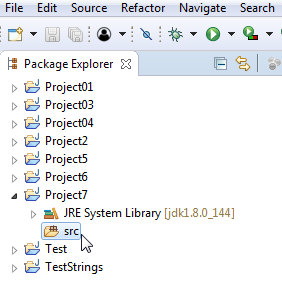
Note: An interactive program is a program that prompts the user to input data

**Task01:** Using Eclipse, write a Java program that displays the message:

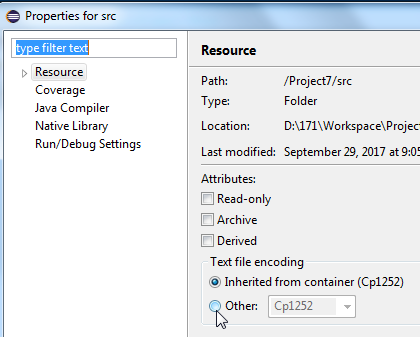
لَا إِلَهَ إِلَّا اللَّهُ وَحْدَهُ لَا شَرِيكَ لَهُ لَهُ الْمُلْكُ وَلَهُ الْحَمْدُ وَهُوَ عَلَى كُلِّ شَيْءٍ قَدِيرٌ

Note: To be able to display Arabic text, your Java file must be encoded in UTF-8. One way of doing that is to use the following steps:

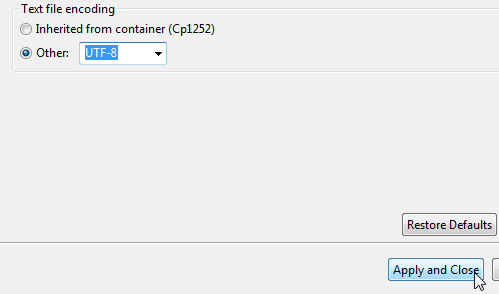
1. Create a new Java project.
2. Right-click on the project’s source folder in the Package Explorer:



1. In the pop-up menu that is displayed, click **Properties**, and then change Text file encoding from Cp1252 to UTF-8



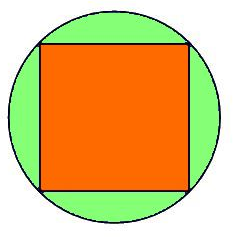
1. Click the **Apply and Close** button:



1. Create a class and paste or write the Arabic text as a string in an output statement like System.out.println
2. Execute your program.

**Task02:** An orange square is inscribed inside a circle as shown in the diagram below. Write an interactive Java program that prompts for and reads the area of the square in square centimeters. It then calculates and prints the area of the green part in square centimeters. Your program must catch **java.util.InputMismatchException** and display an appropriate error message in case such an Exception is thrown.

**Note**: Your program must not use selection statements.



Sample program runs:

|  |  |
| --- | --- |
|  |  |
|  |  |

**Note**: The last output error is due to passing a negative number to the square root method. This error can be avoided by using selection statement. We will cover selection statements later on in the course.

**Task03:** A number of camels are to be distributed equally to each person among a group of people. The distribution is only done if the number of camels is equal to or greater than the number of people. Write an interactive Java program that prompts for and reads the number of camels and the number of people. It then calculates and displays the number of camels each person will get and the number of remaining camels. Your program must catch **java.util.InputMismatchException** and **java.lang.ArithmeticException** and display an appropriate error message should any these Exceptions be thrown.

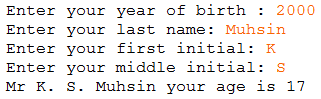
**Note**: Your program must not use selection statements.

Sample program runs:

|  |
| --- |
|  |
|  |
|  |
|  |
|  |
|  |

**Note**: The last output error is due to entering a negative number for the number of camels. This error can be avoided by using selection statement. We will cover selection statements later on in the course.

**Task04:** Write an interactive Java program that prompts for and reads for a person: the year of birth, the last name, and his first and middle initials. It then computes and displays the person’s age in the format shown in the sample program run below:



Note: Your program must declare and use a constant **CURRENT\_YEAR**.

**Task05:** The retirement benefit of an employee can either be calculated as:

CurrentBaseSalary \* NumberOfCompleteYearsOfService.

Or

CurrentBaseSalary \* NumberOfActualYearsOfService.

Write an interactive Java program that prompts for and reads the current base salary of an employee and the actual number of years he has served (**a double value**). It then computes and prints the retirement benefit of the employee by using both the above formulas.

Note:

* The **NumberOfCompleteYearsOfService** used in the above formula is an **integer** value that must be computed by your program; it must not be read.
* The output of your program must be similar to the sample program runs below

|  |
| --- |
|  |
|  |
|  |

**Task 06:** The probability **p** that in a group of **n** people, there are at least two people who have the same birthday is given by the formula:

**p =**

where n is the number of people.

Write an interactive Java program that prompts for and reads the probability **p,** in the interval [0, 1), it then calculates and displays the value of **n**.

**Hint**: Use the larger root of the following quadratic equation:

**n2 – n – 2log10(1 – p)/ log10(364.0/365.0) = 0**

Note: The larger root **r** of the quadratic equation **ax2 + bx + c = 0** is given by:

Sample program runs:

|  |
| --- |
|  |
|  |
|  |
|  |
|  |
|  |
|  |