EECS 1720
Building Interactive Systems

Lecture 3 :: Java Classes & Objects (2)

YORK UNIVERSITÉ UNIVERSITY

Note on installing/using java from terminal (already installed in remote lab)

- Install videos (Java JDK & Eclipse), linked from **lab0.pdf**
- Will need JDK to run java or javac (jvm & compiler) from terminal
  - Test in terminal :
    - type "`java –version`" OR "`javac –version`"
      - If you see a version you are done
  - After creating a project, to run from terminal, navigate to the folder that holds the *.class file:
    - i.e. type: "`java ClassName`"
      - If using a single folder (for source and class files), then it will be the project name (a folder inside your workspace folder)
      - If using separate folders (for source and class files), then the class files will be inside the "bin" sub-directory of your project folder
    - If your file was created inside a package
      - i.e. you see the keyword package at the top of your source file), then you must preface the ClassName with the package name:
      - i.e. type: "`java packageName.ClassName`" to run (from the project folder)

Recall

- Java code is organized fundamentally into files that coincide with “classes” (X.java file has a class called X)
  - Multiple *.java files (multiple classes) can be grouped into a single package (uses package keyword)

- Java classes breakdown into sub-sections
  - (fields, constructors, methods)

    ** constructors are optional

- Two main types of java classes
  - Utility (cannot be instantiated – i.e. cannot create objects)
    - Usually used to group related constants &/or methods
  - Dynamic (can be instantiated – i.e. can create objects)

YORK U
UNIVERSITÉ
UNIVERSITY

Simple utility class structure

Walkthrough Example

YORK UNIVERSITÉ UNIVERSITY

- Basic Utility Classes
  - ALL components (fields & methods) are “static”
  - A basic class is a utility class
    - one static method only – main()
    - if variables are declared as class fields, they must be static
    - static methods only have access to static fields
    - Example 1 (StringUtils.java)
      - Tokenizing a String (via split)
      - Displaying tokens
  - Interaction (101) – getting text-based input into your program
    - String[] args => Command Line input (pre-tokenized)
    - Wrapper Classes & Containers (ArrayList revisited)
    - The Scanner Class
      - Scanner with Strings
      - Scanner with Keyboard Input
      - Scanner with Files (later)

YORK U
UNIVERSITÉ
UNIVERSITY

Example: StringUtils.java

- Recall tokenizing a string…
  - In processing: use split() method
  - In java: split() is a method defined in the String class
    in general, this means it is invoked on a target string

String Functions

`join()`	Combines an array of `Strings` into one `String`, each separated by the character(s) used for the `separator` parameter
`matchAll()`	This function is used to apply a regular expression to a piece of text
`match()`	The function is used to apply a regular expression to a piece of text, and return matching groups (elements found inside parentheses) as a `String` array
`nf()`	Utility function for formatting numbers into strings
`nfc()`	Utility function for formatting numbers into strings and placing appropriate commas to mark units of 1000
`nfp()`	Utility function for formatting numbers into strings
`nfs()`	Utility function for formatting numbers into strings
`splitTokens()`	The `splitTokens()` function splits a `String` at one or many character "tokens"
`split()`	The `split()` function breaks a string into pieces using a character or string as the divider
`trim()`	Removes whitespace characters from the beginning and end of a `String`

Recall: tokenizing a String (in processing)

```
// IDEA:  break up a String into substrings, according to
// some predetermined delimiter (separating character)
// use “split” method:

String str = “input string 4.5 with words and 32 numbers”;
String[] tokens = split(str, “ “);   // delimiter = “ “
// now tokens is an array of Strings:

println(tokens[0]);            // prints “input”
println(tokens[1]);            // prints “string”
println(tokens[2]);            // prints “4.5”
println(tokens[3]);            // prints “with”
…
println(tokens[7]);            // prints “numbers”
```

(some) common String methods (java api)

int	**lastIndexOf**(int ch) Returns the index within this string of the last occurrence of the specified character.
int	**lastIndexOf**(int ch, int fromIndex) Returns the index within this string of the last occurrence of the specified character, searching backward starting at the specified index.
int	**lastIndexOf**(**String** str) Returns the index within this string of the last occurrence of the specified substring.
int	**lastIndexOf**(**String** str, int fromIndex) Returns the index within this string of the last occurrence of the specified substring, searching backward starting at the specified index.
int	**length**() Returns the length of this string.
String	**substring**(int beginIndex) Returns a string that is a substring of this string.
String	**substring**(int beginIndex, int endIndex) Returns a string that is a substring of this string.
char[]	**toCharArray**() Converts this string to a new character array.
String	**toLowerCase**() Converts all of the characters in this String to lower case using the rules of the default locale.
String[]	**split**(**String** regex) Splits this string around matches of the given **regular expression**.
String[]	**split**(**String** regex, int limit) Splits this string around matches of the given **regular expression**.

YORK UNIVERSITÉ UNIVERSITY

(some) common String methods (java api)

split

```
public String[] split(String regex)
```

Splits this string around matches of the given regular expression.

This method works as if by invoking the two-argument `split` method with the given expression and a limit argument of zero. Trailing empty strings are therefore not included in the resulting array.

The string `"boo:and:foo"`, for example, yields the following results with these expressions:

Regex	Result
`:`	`{ "boo", "and", "foo" }`
`o`	`{ "b", "", ":and:f" }`

Parameters:

`regex` - the delimiting regular expression

Returns:

the array of strings computed by splitting this string around matches of the given regular expression

Throws:

`PatternSyntaxException` - if the regular expression's syntax is invalid

`String[]`	`split(String regex)` Splits this string around matches of the given **regular expression**.
`String[]`	`split(String regex, int limit)` Splits this string around matches of the given **regular expression**.

YORK UNIVERSITÉ UNIVERSITY

Recall: tokenizing a String (in java)

```
// IDEA:  break up a String into substrings, according to
// some predetermined delimiter (separating character)
// use “split” method:

String str = “input string 4.5 with words and 32 numbers”;
String[] tokens = str.split(“ “);   // delimiter = “ “
// now tokens is an array of Strings:

println(tokens[0]);            // prints “input”
println(tokens[1]);            // prints “string”
println(tokens[2]);            // prints “4.5”
println(tokens[3]);            // prints “with”
…
println(tokens[7]);            // prints “numbers”
```

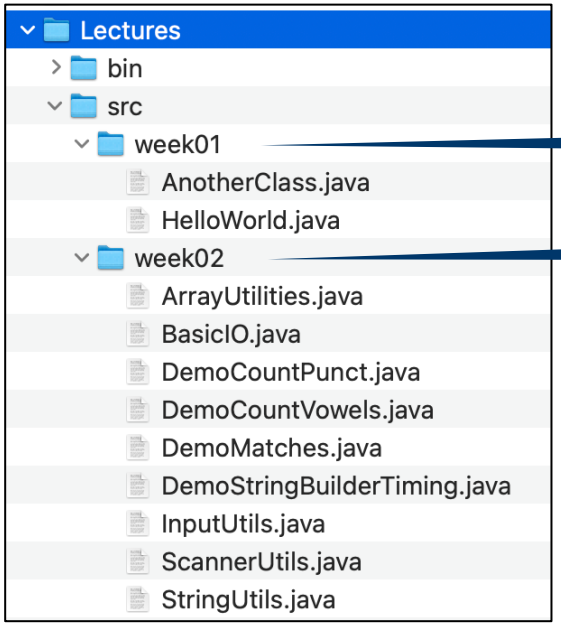
```java
/* A simple class that groups together some string manipulation methods */

public class StringUtils {

    // METHOD (tokenizeString)
    public static String[] tokenizeString(String input) {

        System.out.println("version1");
        System.out.println("--------");
        String[] tokens = input.split(" "); // split based on delimiter = " " (space character)
        return tokens;
    }

    public static void showTokens(String[] tokens) {
        System.out.println("tokens.length = " + tokens.length); // display how many tokens
        System.out.println("----");

        // traditional for loop
        for (int i=0; i<tokens.length; i++) {
          System.out.println("tokens[" + i + "] = " + tokens[i]); // display each token in tokens
        }
        System.out.println("----");

        // // enhanced for loop (works for arrays and collections)
        // int e=0;
        // for (String element : tokens) {
        // System.out.println("tokens[" + e + "] = " + element);
        // e++;
        // }
    }
    // …
```

```java
    // …


    // MAIN
    public static void main(String[] args) {

        // sample string inputs
        String in1 = "150, 302 ;-250, 122 ";
        String in2 = "one flew over the cuckoo’s nest";
        String in3 = "method val1 val2 val3 val4 val5";

        // invoking is possible without class name (since in scope, and static)
        // -> if not static, can only access via an instantiation of the class (object)
        // -> if static generally access through class name (not needed from within same class

        String[] myTokens1 = tokenizeString(in3);                // both invocations work
        String[] myTokens2 = StringUtils.tokenizeString(in3);

        showTokens(myTokens1);
        StringUtils.showTokens(myTokens1);

    }
}
```

static methods always invoked via the classname or without any qualifier (if invoked within the class in which they are defined)

Qualifiers..

- Essentially defines the path to where the class actually lives (i.e. the packages/subpackages)



Project folder (inside workspace folder)

week01 package folder

week02 package folder

HelloWorld class is located at: week01.HelloWorld

StringUtils class is located at: week02.StringUtils

classes in one package do not have access to classes in another package (without doing an import)

Want to call StringUtils methods from HelloWorld ?

```java
package week01;

import week02.StringUtils;

public class HelloWorld {

  public static void main(String[] args) {

    System.out.println("Hello EECS1720 World");

    StringUtils.main(args);

  }

}
```

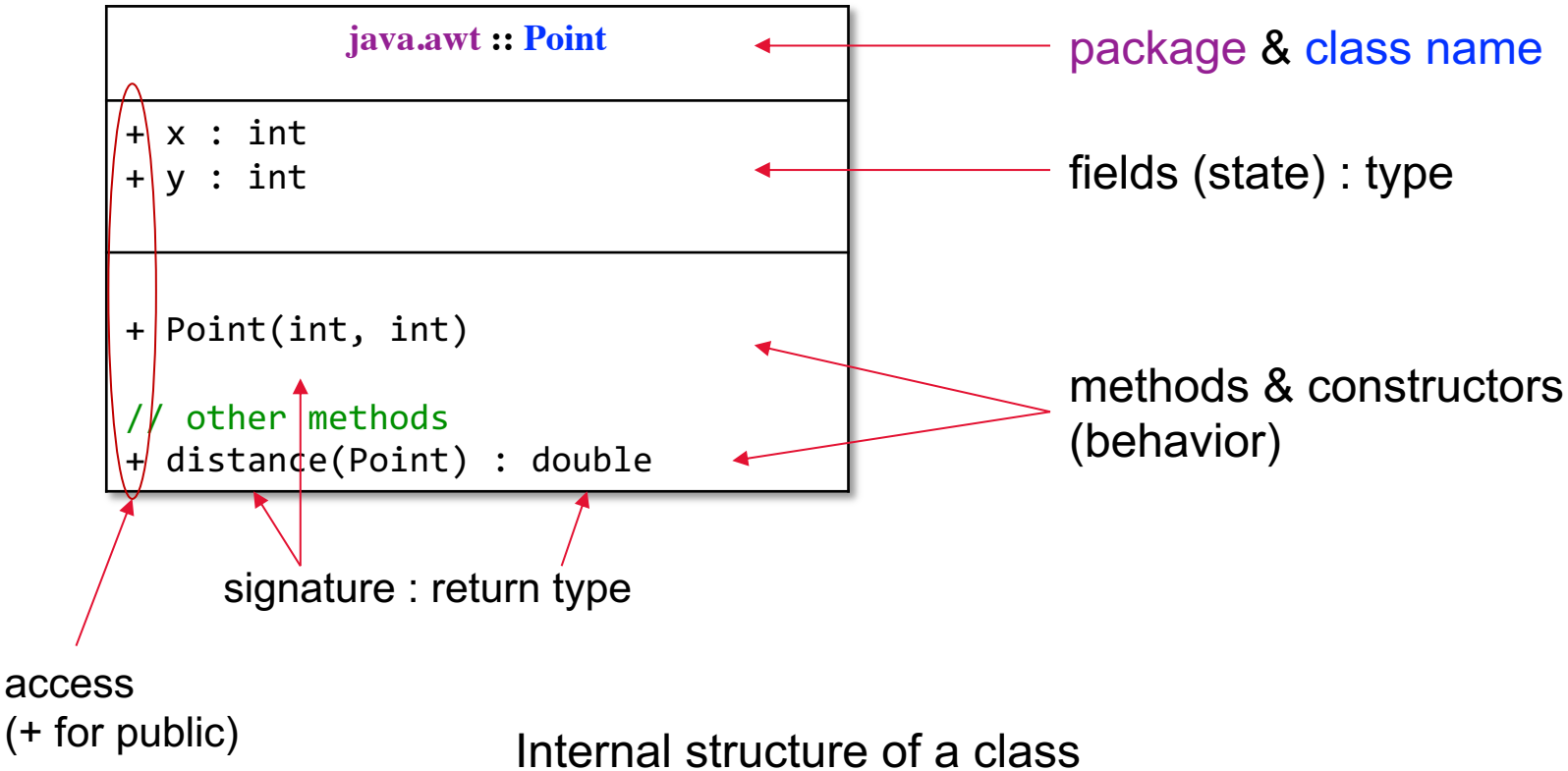
```java
package week02;


public class StringUtils {

  // methods not shown

  public static void main(String[] args) {

    // not shown
  }


}
```

StringUtils and its methods are now accessible from HelloWorld

YORK UNIVERSITÉ UNIVERSITY

UML – universal modeling language

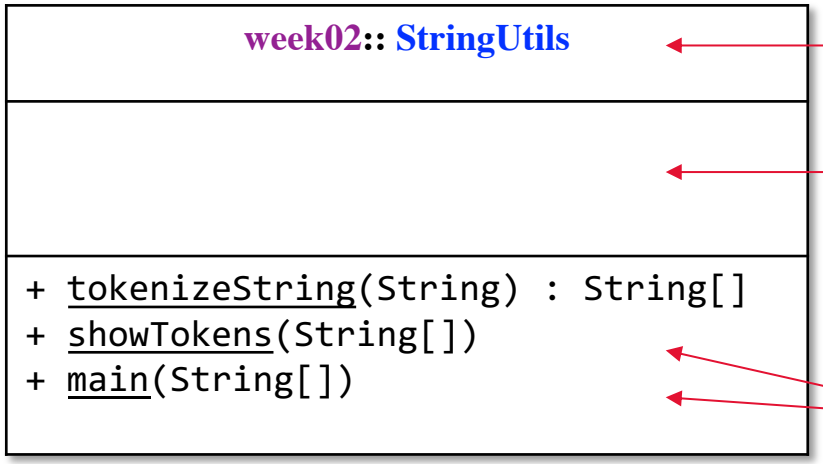
(formal diagrams to represent structure of an application, its components and how it functions, and more → we are primarily interested in <u>class</u> diagrams)



Internal structure of a class

UML – our StringUtils class (so far)

(formal diagrams to represent structure of an application, its components and how it functions, and more → we are primarily interested in class diagrams)



package & class name

fields (state) : type

No fields

methods & constructors (behavior)

static methods are indicated with underlines

** *main usually not shown*

void is considered no return type (so not shown)

```java
public class StringUtils {

    // METHOD (tokenizeString)
    public static String[] tokenizeString(String input) {

        System.out.println("version1");
        System.out.println("--------");
        String[] tokens = input.split(" "); // split based on delimiter = " " (space character)
        return tokens;
    }

    // VERSION 2
    public static String[] tokenizeString(String input, char delimiter ) {

        System.out.println("version2");
        System.out.println("--------");
        String[] tokens = input.split("" + delimiter); // use a char delimiter
        return tokens;
    }
    // VERSION 3
    public static String[] tokenizeString(String input, String delimiter) {
        System.out.println("version3");
        System.out.println("--------");
        String[] tokens = input.split(delimiter); // use a string delimiter
        return tokens;
    }
}
```

UML – our StringUtils class (so far)

(formal diagrams to represent structure of an application, its components and how it functions, and more → we are primarily interested in class diagrams)

week02:: StringUtils

```
+ tokenizeString(String)          : String[]
+ tokenizeString(String, char)    : String[]
+ tokenizeString(String, String)  : String[]

+ showTokens(String[])

+ main(String[])
```

This method is "overloaded"

i.e. multiple versions (must have diff signatures)

YORK UNIVERSITÉ UNIVERSITY

Getting input from Command Line

YORK UNIVERSITÉ UNIVERSITY

Command line arguments ?

- The main method accepts arguments from the command line (or from a run configuration in eclipse) through a String array:

```
public static void main (String[] args) {    }
```

This is a ***pre-tokenized*** version of any text that is added to the command used to launch/run your program

i.e. if your class is called "`MyClass`", and it has a main(), you can run a compiled version of this `MyClass.java` file in the terminal (command line) as:

```
java MyClass                        <= normal way to run java program
java MyClass arg1 arg2 arg3 …       <= OR.. running with input arguments
```

Command line arguments

```
public class MyClass {

  public static void main (String [] args) {



  }

}
```

args→ empty String[ ]

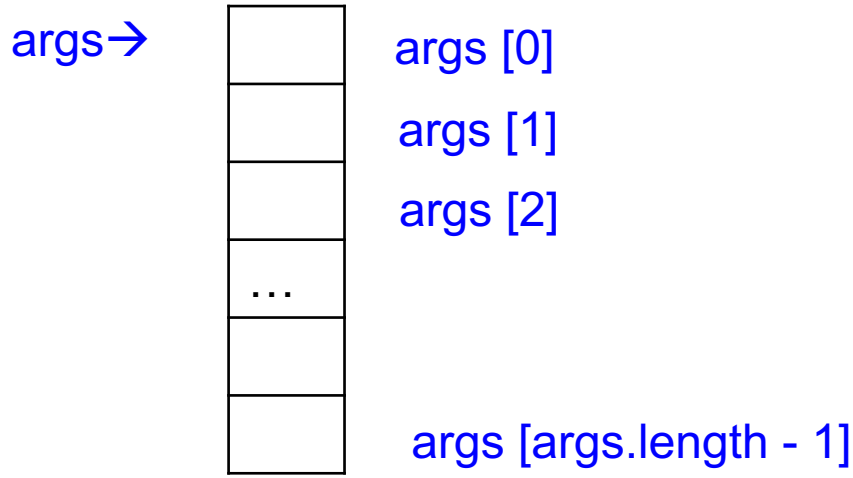
String of “arguments” that can be passed at runtime when program is started

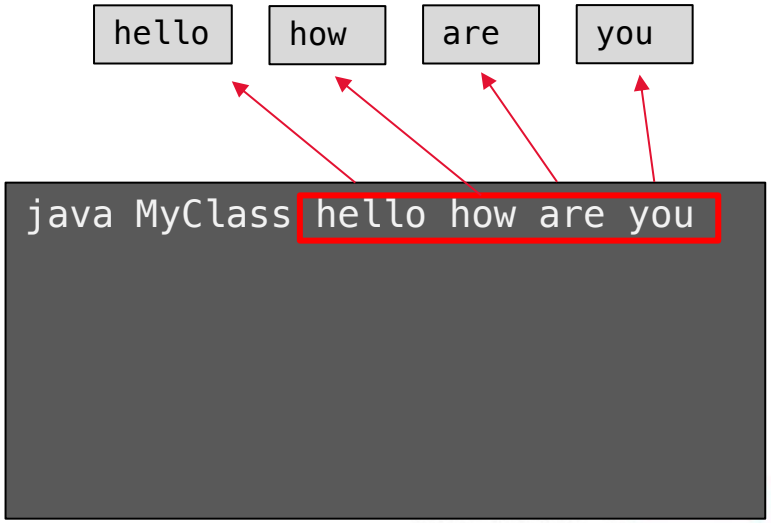
running from terminal

```
java MyClass
```

Command line arguments

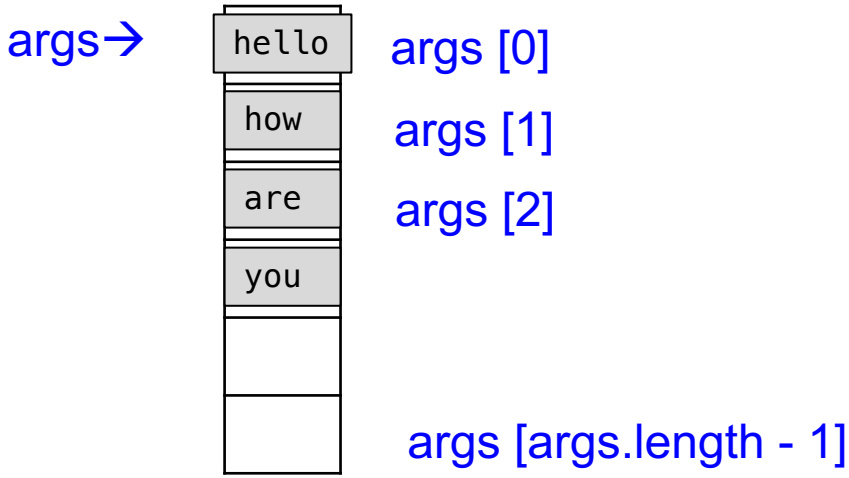
```
public class MyClass {

  public static void main (String [] args) {



  }

}
```





Command line arguments

```
public class MyClass {

  public static void main (String [] args) {



  }

}
```




```
java MyClass hello how are you
```

Example: InputUtils.java

```java
public class InputUtils {

    public static void showCommandLineIn(String[] args) {

        System.out.println("\nCommand Line Input:");
        System.out.println("===================");
        System.out.println("You entered " + args.length + " arguments:");

        // concatenates args back into a single string
        String commandLine = "";

        for (String s : args) {
          commandLine = commandLine + " " + s;     // commandLine += (" " + s);
        }

        System.out.print("input entered was: " + commandLine + "\n\n");

    }

    public static void main(String[] args) {

        showCommandLineIn(args);
        if (args.length==1)
        testForPalindrome(args);
    }

}
```

Getting & processing user (text) inputs at runtime

YORK UNIVERSITÉ UNIVERSITY

Capturing user inputs at run-time

1. Using a class called “Scanner” → from java.util

2. Using “wrapper” classes to ”parse” inputs and convert to values of a certain type

YORK U
UNIVERSITÉ
UNIVERSITY

Scanner class API (constructors)

Constructors

Constructor and Description
`Scanner(File source)` Constructs a new Scanner that produces values scanned from the specified file.
`Scanner(File source, String charsetName)` Constructs a new Scanner that produces values scanned from the specified file.
`Scanner(InputStream source)` Constructs a new Scanner that produces values scanned from the specified input stream.
`Scanner(InputStream source, String charsetName)` Constructs a new Scanner that produces values scanned from the specified input stream.
`Scanner(Path source)` Constructs a new Scanner that produces values scanned from the specified file.
`Scanner(Path source, String charsetName)` Constructs a new Scanner that produces values scanned from the specified file.
`Scanner(Readable source)` Constructs a new Scanner that produces values scanned from the specified source.
`Scanner(ReadableByteChannel source)` Constructs a new Scanner that produces values scanned from the specified channel.
`Scanner(ReadableByteChannel source, String charsetName)` Constructs a new Scanner that produces values scanned from the specified channel.
`Scanner(String source)` Constructs a new Scanner that produces values scanned from the specified string.

Scanner can be used with many different types of inputs. For e.g:

- String
- InputStream
- File

YORK UNIVERSITÉ UNIVERSITY

Scanner (partial API) :: *java.util.Scanner*

boolean	**hasNext**() Returns true if this scanner has another token in its input.
String	**next**() Finds and returns the next complete token from this scanner.
double	**nextDouble**() Scans the next token of the input as a `double`.
float	**nextFloat**() Scans the next token of the input as a `float`.
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.

Scanner objects have associated methods to extract & parse portions of an input (sequentially & in chunks)

YORK UNIVERSITÉ UNIVERSITY

Example: using Scanner on a String input

```
String input = “Once upon a time”;

Scanner strScan = new Scanner(input);        // Scanner class is dynamic
                                             // => can create objects

String token;

token = strScan.next();            // token = “Once”
token = strScan.next();            // token = “upon”
token = strScan.next();            // token = “a”
token = strScan.next();            // token = “time”
```

- Scanner object (strScan) tracks where in the input string (input) it currently is, and scans through chunks of characters sequentially …
  - methods modify Scanner’s position in the input string, return next chunk (token)… and may interpret the chunk
    - next() returns the next sequence of characters until next whitespace - interprets chunk as a String
    - nextLine() gets all characters until next newline character – also interprets chunk as a String

YORK UNIVERSITÉ UNIVERSITY

Example: using Scanner on a String input

```
String input2 = “14 50.1231”;

Scanner strScan = new Scanner(input2);          // this Scanner object is connected to
                                                // a different string (thus diff object)

int token1;
double token2;

token1 = strScan.nextInt();                     // token = 14
token2 = strScan.nextDouble();                  // token = 50.1231
```

- nextInt() gets next chunk and interprets it as an int
- nextDouble() gets next chunk and interprets it as a double

- *Scanner advances (scans) along string (or consumes string) with every next() type call*
  - *Think of it as a window that moves further along the string with each next___() method call*

Can also leverage “Wrapper” classes

- primitive data types have a set of non-primitive counterparts (similar, but with constructors & methods)

- int ←→ Integer
- double ←→ Double
- long ←→ Long
- boolean ←→ Boolean
- char ←→ Character

etc…

YORK UNIVERSITÉ UNIVERSITY

Wrappers have both static/non-static features

- Non-static:
  - Can create objects using constructors:

Constructors

Constructor and Description
`Integer(int value)` Constructs a newly allocated `Integer` object that represents the specified `int` value.
`Integer(String s)` Constructs a newly allocated `Integer` object that represents the `int` value indicated by the `String` parameter.

  - E.g.

```
Integer myInt1 = new Integer(5);
Integer myInt2 = new Integer("1710");
```

- Static:
  - Some static (class invoked) methods to parse strings into primitive types

`static int`	`parseInt(String s)` Parses the string argument as a signed decimal integer.

  - E.g.

```
int myInt3 = Integer.parseInt("1720");
```

Wrapper class methods for parsing strings

```
String strVal = /* not shown */;

int iVal = Integer.parseInt(strVal);
short sVal = Short.parseShort(strVal);
long lVal = Long.parseLong(strVal);

float fVal = Float.parseFloat(strVal);
double dVal = Double.parseDouble(strVal);
```

- Mostly see these parse methods in numeric types
- Can be used on string tokens extracted using Scanner

YORK UNIVERSITÉ UNIVERSITY

Example

```
// assume string of values (space separated): “x1 y1 x2 y2”
// representing two points (end points of a line in x,y)
String linePoints = “14.5 -3.74 100.242 61”;


Scanner in = new Scanner(linePoints);

double x1 = Double.parseInt(in.next());   // x1==14.5
double y1 = Double.parseInt(in.next());   // y1==-3.74
double x2 = Double.parseInt(in.next());   // x2==100.242
double y2 = Double.parseInt(in.next());   // y2==61.0
```

Using Scanner to capture Keyboard Input

- Connecting to Keyboard Input (System.in)

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

- Parsing inputs typed in by user at run time:

```
System.out.println("Enter a value between 50 and 100: ");
int val1 = in.nextInt();
int val2 = in.nextInt();
```

OR

```
System.out.println("Enter a value between 50 and 100: ");
int val1 = Integer.parseInt(in.next());
int val2 = Integer.parseInt(in.next());
```

YORK U
UNIVERSITÉ
UNIVERSITY

What happens if parse or next methods run to interpret, but don't encounter valid string?

- E.g.

```
String str = "1es.14";

Scanner in = new Scanner(str);
int value = in.nextInt();

// lets look at API:
```

parseInt

```
public static int parseInt(String s)
                    throws NumberFormatException
```

Parses the string argument as a signed decimal integer. The characters in the string must all be decimal digits, except that the first character may be an ASCII minus sign '-' ('\u002D') to indicate a negative value or an ASCII plus sign '+' ('\u002B') to indicate a positive value. The resulting integer value is returned, exactly as if the argument and the radix 10 were given as arguments to the parseInt(java.lang.String, int) method.

Parameters:

s - a String containing the int representation to be parsed

Returns:

the integer value represented by the argument in decimal.

Throws:

NumberFormatException - if the string does not contain a parsable integer.

Other useful Scanner methods (… later)

- Can ”look” ahead (before scanning)

	nextByte() method.
boolean	**hasNextDouble**() Returns true if the next token in this scanner's input can be interpreted as a double value using the **nextDouble()** method.
boolean	**hasNextFloat**() Returns true if the next token in this scanner's input can be interpreted as a float value using the **nextFloat()** method.
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.
boolean	**hasNextInt**(int radix) Returns true if the next token in this scanner's input can be interpreted as an int value in the specified radix using the **nextInt()** method.
boolean	**hasNextLine**() Returns true if there is another line in the input of this scanner.
boolean	**hasNextLong**() Returns true if the next token in this scanner's input can be interpreted as a long value in the default radix using the **nextLong()** method.

Example (checking before reading)

```
import java.util.Scanner;


public class GuessingGame {

    public static void main(String[] args) {

        Scanner in = new Scanner(System.in);

        System.out.println(“.. can you guess the number? ”);
        double guess;

        if (in.hasNextDouble()) {
          guess = in.nextDouble();
          System.out.println(“you entered: “ + guess);
        }

        System.out.println(“program is ending now”);
    }
}
```

More Detailed Example: parsing input using ArrayLists…

- Let's say we want to process a string that has a mixture of doubles and ints, and we want to count and extract them (while ignoring all the other stuff)

ECLIPSE DEMO

Recall:
Arrays are a bit confusing
ArrayLists are much easier to work with

- Arrays have one field (length), but no methods

```
String[] myStringArray = new String[10]; // fixed size
myStringArray[0] = “eecs1710”;
myStringArray[1] = “eecs1720”;
out.println(“length = ” + myStringArray.length);
```

- ArrayLists are a reference type by design, thus support fields and methods

```
ArrayList<String> myArrayList = new ArrayList<String>(); // can grow

// no length field, only size() method
out.println(myArrayList.size());              // empty, so size = 0

myArrayList.add(“eecs1710”);
myArrayList.add(“eecs1720”);

out.println(myArrayList.size());
```

Note on ArrayLists

- Must always use a Reference type as the element <>
  - Cannot use primitive types

- ArrayList<Integer>
- ArrayList<Double>
- etc

- Why? ArrayList makes use of certain methods in these classes to function correctly
  - E.g. to compare elements (e.g. equals() method is needed), also when searching for an element in the collection..

YORK UNIVERSITÉ UNIVERSITY

```java
public static void scanKeyboardInput() {

        Scanner in = new Scanner(System.in);                                // keyboard
        System.out.print("Please enter a set of space separated values: ");  // prompt

        // create some counters
        int countIntegers = 0;
        int countReals = 0;

        // create some storage for numbers found (containers must use wrapper classes)
        ArrayList<Integer> intList = new ArrayList<Integer>();
        ArrayList<Double> realList = new ArrayList<Double>();

        // grab entire line of input from keyboard, make a string based scanner
        Scanner line = new Scanner(in.nextLine());

        while (line.hasNext()) {

                // look ahead
                if (line.hasNextInt()) {
                        countIntegers++;
                        intList.add(line.nextInt());
                }
                else if (line.hasNextDouble()) {
                        countReals++;
                        realList.add(line.nextDouble());
                }
                else {
                        line.next();
                }
        }
        line.close();         // good practice to close scanner objects when done
        in.close();

        System.out.println("found " + countIntegers + " integers, and " + countReals + " reals");
        System.out.println("\nints:  \n" + intList.toString());
        System.out.println("\nreals: \n" + realList.toString());

}
```

```
Please enter a set of space separated values: 234.5235 4.23 5252 344 6 2 r4g5 glsdkjh &%&# 32 4.522 0.342 -56
found 6 integers, and 4 reals

ints:
[5252, 344, 6, 2, 32, -56]

reals:
[234.5235, 4.23, 4.522, 0.342]
```

Takeaways

- Utility classes have only static components (fields/methods)
- Methods must be invoked using the class name
  - can be invoked directly if invoking from a method within the same class
  - Must invoke using full qualifier (package.classname) if class is in a different package
- UML diagrams can be used to give a quick overview of the methods, fields and constructors in a class
- User input can make its way into a java program from:
  - Command line arguments
  - Keyboard text (typed at runtime) => via System.in
- Scanner & wrapper classes are useful for reading in user input text, and processing them (converting to useable variables)

To do (practice)

- Write a static method that prompts for and reads an integer from the user and then outputs the integer's absolute value.
  - *Do not use Math.abs*.
- Write a static method that prompts a user for three double values and then tests if the the third value is within the range of the first two
  - You may use Math.min() and Math.max() to do this
- Explore the API for the numeric wrapper classes (in java.lang)
  - Integer, Short, Long, Float, Double
- Explore the API for Scanner class (in java.util)
  - Look at available methods, constructors

YORK U
UNIVERSITÉ
UNIVERSITY