Universidad
Carlos III de Madrid
www.uc3m.es

Lesson 4

Utility classes: Math, String, I/O

Programming

Grade in Computer Engineering

Outline

1. Math class
2. Wrapper classes
3. String management
4. Input and output
    - Printing on the screen
    - Reading from keyboard
    - Arguments to `main`
5. Collections

Outline

1. **Math class**
2. Wrapper classes
3. String management
4. Input and output
   - Printing on the screen
   - Reading from keyboard
   - Arguments to `main`
5. Collections

`Math` is a special class in JDK that contains mathematic functions and constants, all static.

Attributes:

- E and PI

Functions:

- Absolute value
- Trigonometric
- Maximum and minimum of two numbers
- Natural logarithms and base 10
- Exponential
- Square and cubic roots
- Random values

http://java.sun.com/javase/7/docs/api/java/lang/Math.html

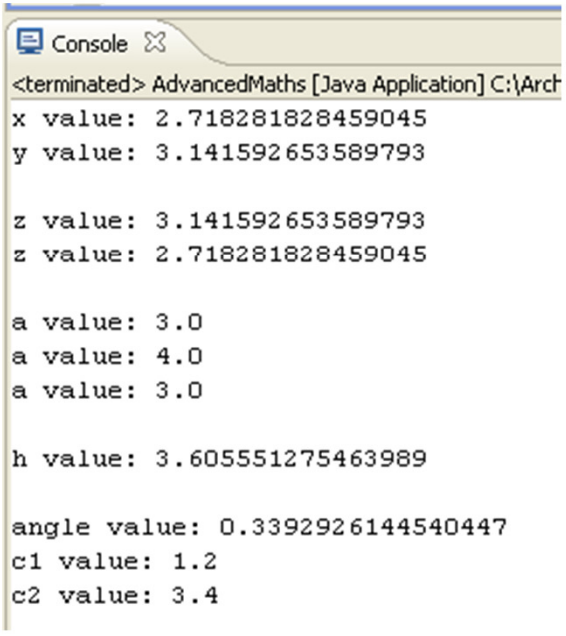
`Math` class is inside the `java.lang` package – **it is not necessary to import it!**

Other Java mathematical libraries: *Apache Jakarta Math library*

1. Math library

```java
/* Advanced math operators */
double x, y, z;

// Values
x = Math.E;
System.out.println("x value: " + x);
y = Math.PI;
System.out.println("y value: " + y);

// Comparisons
System.out.println("");
z = Math.max(x, y);
System.out.println("z value: " + z);
z = Math.min(x, y);
System.out.println("z value: " + z);

// Rounding
double a;
System.out.println("");
a = Math.round(y);
System.out.println("a value: " + a);
a = Math.ceil(y);
System.out.println("a value: " + a);
a = Math.floor(y);
System.out.println("a value: " + a);

// Exponentiation
double c1, c2, h;
System.out.println("");
c1 = 1.2;
c2 = 3.4;
h  = Math.sqrt( Math.pow(c1, 2) + Math.pow(c2, 2) );
System.out.println("h value: " + h);
```

```java
// Trigonometric
double tangent, angle, c1_fromSine, c2_fromCosine;
System.out.println("");
tangent = c1/c2;
angle = Math.atan(tangent);
System.out.println("angle value: " + angle);
c1_fromSine = h * Math.sin(angle);
c2_fromCosine = h * Math.cos(angle);
System.out.println("c1 value: " + c1_fromSine);
System.out.println("c2 value: " + c2_fromCosine);
```



AdvancedMaths.java

Outline

1. Math class
2. **Wrapper classes**
3. String management
4. Input and output
   - Printing on the screen
   - Reading from keyboard
   - Arguments to `main`
5. Collections

Primitive datatypes are used most of the time to work with numbers

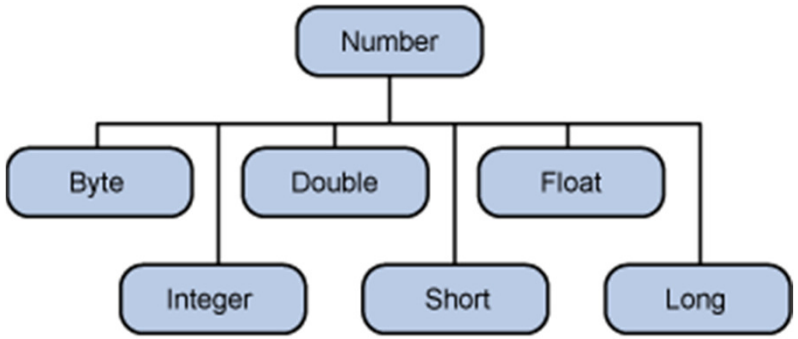
```
int x = 1;
double y = 1.2, z;
z = x * y;
```

Java provides, in addition, *wrapper* classes for each of the primitive data types

- These classes "wrap" the primitive value in an object
- Often, wrapping is done by the compiler
  - If a primitive value is used when an object is expected, the compiler automatically *boxes* the primitive into an object
  - If an object is used when a primitive value is expected, the compiler automatically *unboxes* the object and uses the value

```
Integer x, y;
x = 12;
y = 15;
System.out.println(x+y);
```

Wrapper classes are inside the `java.lang` package – **it is not necessary to import them!**



There are three reasons that you might use a `Number` object rather than a primitive:

- As an argument of a method that expects an object (often used when manipulating collections of numbers).
- To use constants defined by the class, such as `MIN_VALUE` and `MAX_VALUE`, that provide the upper and lower bounds of the data type.
- To use class methods for converting values to and from other primitive types, for converting to and from strings, and for converting between number systems (decimal, octal, hexadecimal, binary)

Large integer values can be managed with the `BigInteger` class

http://download.oracle.com/javase/7/docs/api/java/math/BigInteger.html

```
BigInteger i = new BigInteger("92233720368547758O8");
```

More precise floating point values can be managed with the `BigDecimal` class

http://download.oracle.com/javase/7/docs/api/java/math/BigDecimal.html

```
BigDecimal j = new BigDecimal("10E-500");
```

Extended classes are inside the `java.math` package – **it is necessary to import them!**

No automatic boxing-unboxing (*cf.* wrapper classes)

Outline

1. Math class
2. Wrapper classes
3. **String management**
4. Input and output
   - Printing on the screen
   - Reading from keyboard
   - Arguments to `main`
5. Collections

`String`s are complex data types to represent and manage a string of characters

`String``S` are enclosed between double quotes (`""`)

> The **`String`** class contains static methods to convert numbers to strings

> The wrapping classes (**`Integer`**, **`Float`**, etc.) contain static methods to convert values to **`String`**

`String` ⇨ basic type

byte: `Byte.parseByte(<string>)`

short: `Short.parseShort(<string>)`

int: `Integer.parseInt(<string>)`

long: `Long.parseLong(<string>)`

boolean: `Boolean.parseBoolean(<string>)`

float: `Float.parseFloat(<string>)`

double: `Double.parseDouble(<string>)`

char: `<string>.charAt(<position>)`

`String` ⇨ basic type

If the format of the string is not correct, an error occurs (`NumberFormatException`)

```
int x;
x = Integer.parseInt("abc");  // ERROR
```

To manage conversion errors, use a `try-catch` instruction

```
int x = 0;
try {
  x = Integer.parseInt("abc");
} catch (NumberFormatException e) {
  System.out.println("Wrong number format");
}
```

Basic type ⇨ `String`

`String.valueOf(<number expression>)`

```
byte b;
short s;
```

> byte:

```
String s = String.valueOf(b);
```

> short:

```
String s = String.valueOf(s);
```

This conversion is **automatically performed by the compiler in some cases** where a `String` is expected

```
> System.out.println("Sum: " + (3+5) );
> System.out.println("Sum: " + String.valueOf(3+5) );
```

3. String management
Examples

```
/* Converting strings to numbers */
String d = "18.3";
float  d_float  = Float.parseFloat(d);
double d_double = Double.parseDouble(d);
// int d_int    = Integer.parseInt(d);
System.out.println(d_float);
System.out.println(d_double);

String i = "18";
int    i_int    = Integer.parseInt(i);
short  i_short  = Short.parseShort(i);
long   i_long   = Short.parseShort(i);
float  i_float  = Float.parseFloat(i);
System.out.println(i_int);
System.out.println(i_short);
System.out.println(i_long);
System.out.println(i_float);
System.out.println("---------");

/* Converting numbers to strings */
String s;
s = String.valueOf(d_float);
System.out.println(s);
s = String.valueOf(d_double);
System.out.println(s);
s = String.valueOf(i_int);
System.out.println(s);
s = String.valueOf(i_long);
System.out.println(s);
```

```
Console
<terminated> Strings [Java Application]
18.3
18.3
18
18
18
18.0
---------
18.3
18.3
18
18
```

Strings.java

3. String management
Other string methods

Type	Method
int	***compareTo***(String anotherString) Compares two strings lexicographically
boolean	***contains***(CharSequence s) Returns true if and only if this string contains the specified sequence of char values
boolean	***equals***(Object anObject) Compares this string to the specified object
int	***indexOf***(String str) Returns the index within this string of the first occurrence of the specified substring
int	***lastIndexOf***(String str) Returns the index within this string of the last occurrence of the specified substring
boolean	***matches***(String regex) Tells whether or not this string matches the given regular expression
String	***replaceAll***(String regex, String replacement) Replaces each substring of this string that matches the given regular expression with the given replacement
String []	***split***(String regex) Splits this string around matches of the given regular expression
String	***substring***(int beginIndex, int endIndex) Returns a new string that is a substring of this string.

http://download.oracle.com/javase/7/docs/api/java/lang/String.html

Outline

1. Math class
2. Wrapper classes
3. String management
4. **Input and output**
   - **Printing on the screen**
   - Reading from keyboard
   - Arguments to `main`
5. Collections

`System.out` for printing on screen

Methods for printing on screen:

Without a line jump:

```
System.out.print(<String expression>);
```

With a line jump:

```
System.out.println(<String expression>);
```

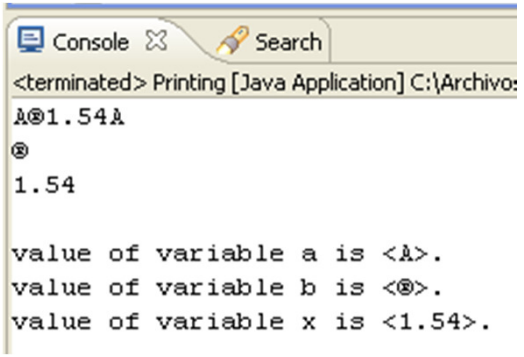
Line jump can be printed with the new line character '\n':

`println("Hi!")` is equivalent to `print("Hi!\n")`

4. Input and output

Printing on the screen

```
char a = 'A';
char b = '\u00AE';
double x = 1.54;

// Without new line
System.out.print(a);
System.out.print(b);
System.out.print(x);

// With new line (after the printing)
System.out.println(a);
System.out.println(b);
System.out.println(x);

// Concatenation of strings
System.out.print("\nvalue of variable a is <" + a + ">.\n");
System.out.println("value of variable b is <" + b + ">.");
System.out.print("value of variable x is <" + x + ">.");
```



Printing.java

System.out.printf

String including conversion characters (*placeholders*)

%d: integers

%f: floating point numbers

%c: characters

%s: strings

…different number notations, dates, etc.

Corresponding expressions (as many as placeholders)

```
System.out.printf("%d", <integer expression>);
System.out.printf("%f %d", <real expression>, <integer expression>);
```

Round floating-point values

```
System.out.printf("%2.1f", <real expression>);
```

2 digits for the integer part, 1 digit for the decimal part

```
System.out.printf("%.3f", <real expression>);
```

All digits in the integer part, 3 digits for the decimal part

Use different notations

```
System.out.printf("%e", <real expression>);
```

Exponential notation

Set field widths

```
System.out.printf("%5d", <integer expression>);
```

5 digits for the integer part

and more…

4. Input and output

Formatting output

```
// Printing integer values
int a=1, b=2, c=3;
System.out.printf("values: %d, %d, %d \n", a, b, c);
System.out.printf("sum: %d \n", a+b+c);
System.out.printf("%5d %5d %5d \n", 0, 1, 2);
System.out.printf("%5d %5d %5d \n", a, b, c);


// Printing floating point values
double x=Math.PI;
System.out.printf("x: %f \n", x);
System.out.printf("x: %1.9f \n", x);
System.out.printf("x: %e \n", x);

// Printing strings and characters
String s = "Hello world!";
System.out.printf("A typical first message in Java is %s \n", s);
System.out.printf("A typical first message in Java is \"%s\" \n", s);
```

```
Console
<terminated> FormattedOutput [Java Application] C:\Program Fil
values: 1, 2, 3
sum: 6
    0     1     2
    1     2     3
x: 3,141593
x: 3,141592654
x: 3.141593e+00
A typical first message in Java is Hello world!
A typical first message in Java is "Hello world!"
```

FormattedOutput.java

Outline

1. Math class
2. Wrapper classes
3. String management
4. **Input and output**
   - Printing on the screen
   - **Reading from keyboard**
   - Arguments to `main`
5. Collections

`Scanner` class can be used to read values from the keyboard **(see Lesson 2, slide 33)**

Use:

1. Import `java.util.* package`

```
import java.util.*;
```

2. Declare and initialize a `Scanner` object `sc`

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

3. Read values

Integer:	`int a = sc.nextInt ();`	
Float:	`float b = sc.nextFloat();`	
Double:	`double c = sc.nextDouble();`	
String:	`String s = sc.next();`	(No blank spaces)
	`String s = sc.nextLine();`	(With blank spaces)

…

Using **streams** to read from the keyboard:

1. Import `java.io.*`
2. Create an `InputStreamReader` *isr* based on `System.in`
3. Create a `BufferedReader` *br* based on *isr*
4. Read a line *s* (`String` type) from the user with `br.readline()` [`try-catch` must be used]
5. Convert *s* to the expected datatype with the parsing methods [`try-catch` should be used]

4. Input and output

Alternative to `Scanner`

```java
import java.io.*;

public class InputStreams {

    public static void main(String[] args) {

        // Prepare objects for reading
        InputStreamReader isr = new InputStreamReader(System.in);
        BufferedReader br = new BufferedReader(isr);

        // Indicate the user what he/she should type
        System.out.println("Enter your name: ");
        String userName = "";

        // Read
        try {
            userName = br.readLine();
        } catch (IOException e) {
            System.out.println("Error while reading name.");
            System.out.println("Finishing the program!");
            System.exit(-1);
        }

        // Say hello
        System.out.println("Hello " + userName + "!");
```

InputStreams.java

4. Input and output

Alternative to `Scanner`

```
        // Read age
        System.out.println("How old are you, " + userName + "?");
        int userAge = 0;
        String userAnswer = "";

        try {
            userAnswer = br.readLine();
            userAge = Integer.parseInt(userAnswer);
        } catch (IOException e) {
            System.out.println("Error while reading age.");
            System.out.println("Finishing the program!");
            System.exit(-1);
        } catch (NumberFormatException e) {
            System.out.println("Age is no correct.");
            System.out.println("(typed value was: " + userAnswer + ").");
            System.out.println("Finishing the program!");
            System.exit(-1);
        }

        // Say polite message
        System.out.println("Only " + userAge + "?");
        System.out.println("You're really young, " + userName + "!");
    }

}
```

InputStreams.java

Outline

1. Math class
2. Wrapper classes
3. String management
4. **Input and output**
   - Printing on the screen
   - Reading from keyboard
   - **Arguments to `main`**
5. Collections

The `main` method receives an array of `String` with the strings given as arguments from the command line

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

These strings can be used in the program

`args[0], args[1], …`

To **use arguments as numbers**, **parsing is required**

The command line arguments are space separated

An argument can span several words if it is enclosed with " "

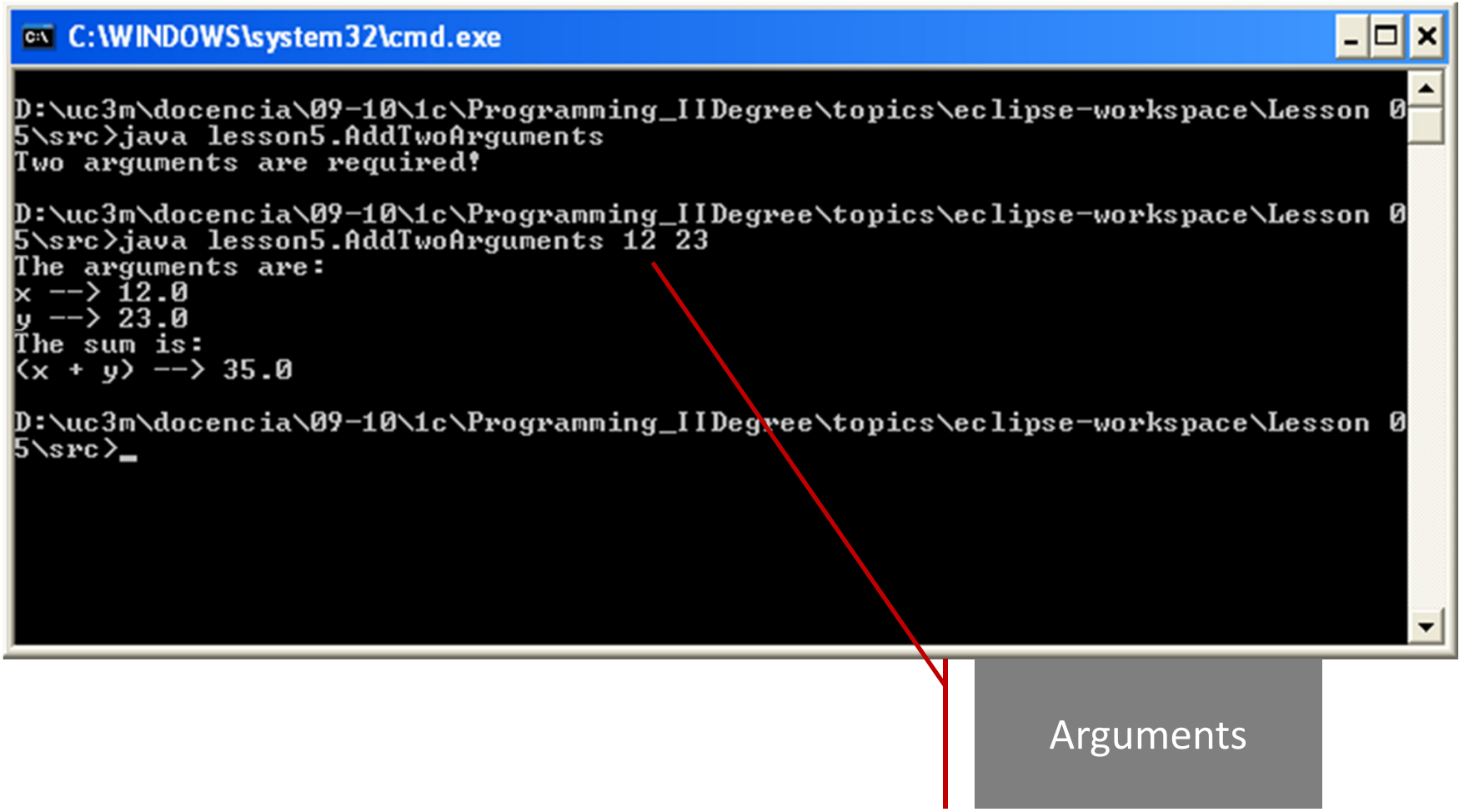
These arguments can be also introduced in Eclipse

Run >> Run configurations --> Java Application, Arguments tab

An argument can span several words if it is enclosed with " "

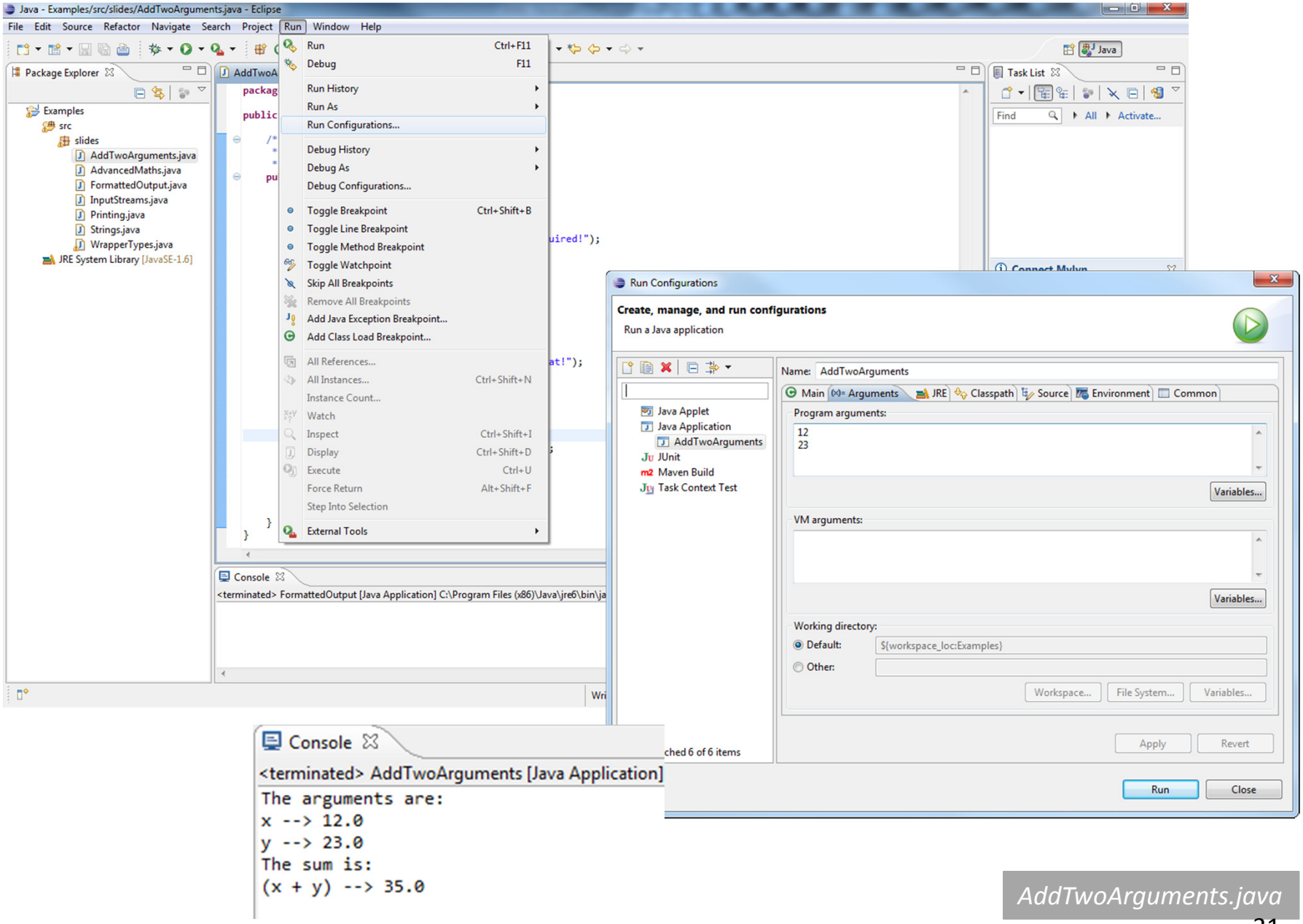
4. Input and output

Arguments to `main`



AddTwoArguments.java

Arguments to `main`



AddTwoArguments.java

4. Input and output

Arguments to `main`

Arguments.java

```
package slides;

public class Arguments {

    /**
     * @param args Command-line arguments
     */
    public static void main(String[] args) {

        // Check if arguments have been passed
        // If not, the program ends
        if(args.length > 0) {
            System.out.println(
                    "You called the program with [" +
                    args.length +
                    "] arguments");

            // Print out all the arguments
            for(int i=0; i<args.length; i++) {
                System.out.println("Argument " + i + " is: " + args[i]);
            }

        } else {
            System.out.println("No arguments, no fun!");
        }

    }

}
```

Arguments.java

4. Input and output

Arguments to `main`

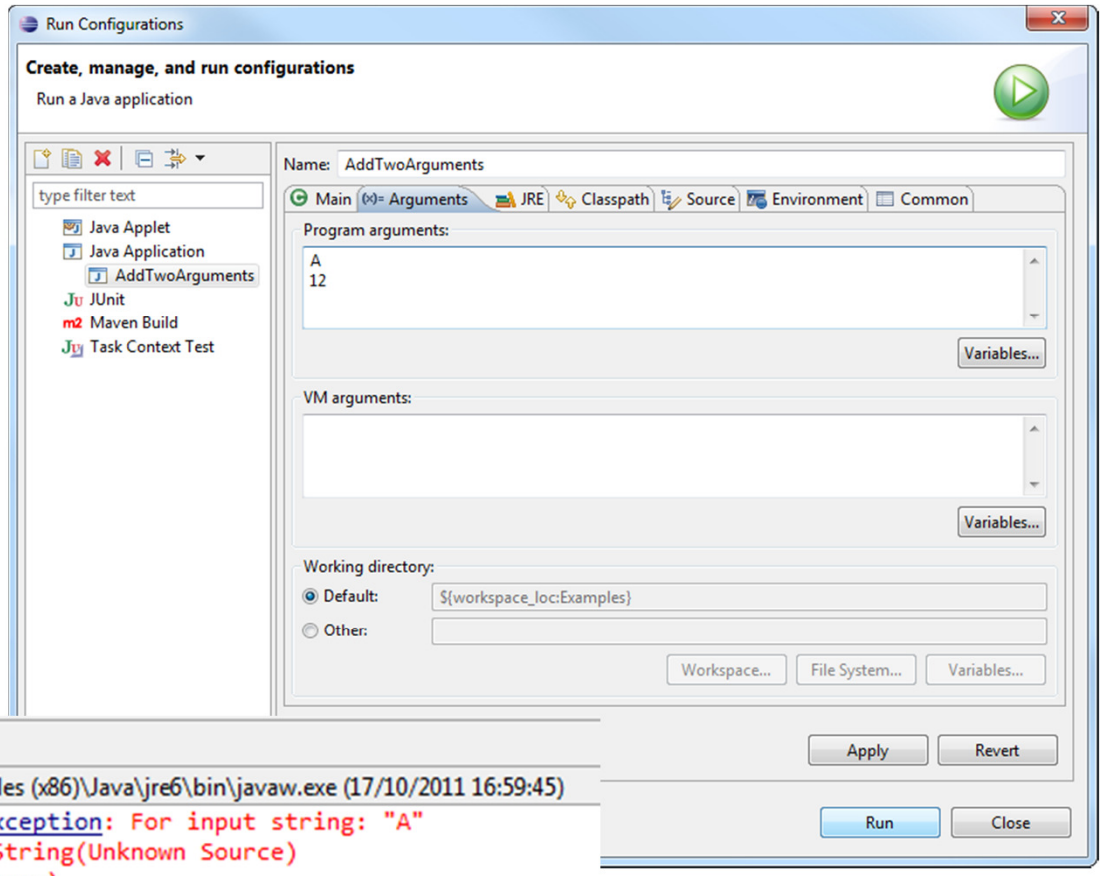
```java
public class AddTwoArguments {

    /**
     * @param args Command-line arguments
     */
    public static void main(String[] args) {

        // Check number of params
        if(args.length != 2) {
            // More or less than 2 arguments
            System.out.println("Two arguments are required!");

        } else {
            // Exactly 2 arguments
            double x = 0, y = 0;

            try {
                x = Double.parseDouble(args[0]);
                y = Double.parseDouble(args[1]);
            } catch (NumberFormatException e) {
                System.out.println("Wrong number format!");
                System.out.println("Ending program");
                System.exit(-1);
            }

            double sum = x + y;

            System.out.println("The arguments are: ");
            System.out.println("x --> " + x);
            System.out.println("y --> " + y);
            System.out.println("The sum is: ");
            System.out.println("(x + y) --> " + sum);
        }
    }
}
```

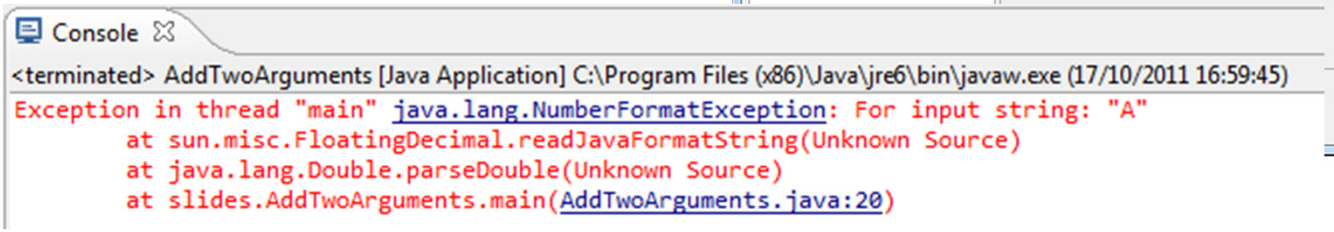
AddTwoArguments.java

4. Input and output

Arguments to `main`

```
// try {
    x = Double.parseDouble(args[0]);
    y = Double.parseDouble(args[1]);
// } catch (NumberFormatException e) {
//   System.out.println("Wrong number format!");
//   System.out.println("Ending program");
//   System.exit(-1);
// }

double sum = x + y;

System.out.println("The arguments are: ");
System.out.println("x --> " + x);
System.out.println("y --> " + y);
System.out.println("The sum is: ");
System.out.println("(x + y) --> " + sum);
```





AddTwoArguments.java

Outline

1. Math class
2. Wrapper classes
3. String management
4. Input and output
    - Printing on the screen
    - Reading from keyboard
    - Arguments to main
5. **Collections**

Arrays:

Hold a sequence of basic type values or objects

They have fixed-length, specified in the initialization with **`new`** (can be consulted with `array.length`)

If we want to add a new element that exceeds its capacity, we need to define a new larger array, and then copy the first one into the second one (`System.arraycopy`)

Dynamic arrays:

Hold objects of a single type (jdk1.5 introduces “generics”, with extended functionalities)

They have variable length and elements can be dynamically added or removed from the collection

Java provides several pre-implemented classes to represent and manage sets/lists in the package `java.util`: `ArrayList`, `Vector`, etc.

ArrayList is a class for arrays that grow dynamically and in which we can store all sorts of elements

Some methods:

`boolean` **`add`**`(Object o)`: Add *o* to the end

`void` **`add`**`(int index, Object o)`: Add *o* to the specified position *index*. The indexes start at zero (the same as arrays)

`Object` **`get`**`(int index)`: Return the element at a specific position. Since it returns a generic *Object*, we need to make a explicit casting to the expected object

`int` **`size`**`()`: Return the size (number of elements) of the vector

`Object` **`remove`**`(int index)`: Deletes the element at position *index*

ArrayListExample.java

```
import java.util.ArrayList;

public class ArrayListExample {

    public static void main(String[] args) {

        ArrayList<Integer> myList;
        myList = new ArrayList<Integer>();

        myList.add(5);
        myList.add(10);
        myList.add(15);

        System.out.println(myList.get(0));
        System.out.println(myList.get(1));
        System.out.println(myList.get(2));

        System.out.println(myList.size());
        myList.remove(0);
        System.out.println(myList.size());

        System.out.println(myList);

    }

}
```

A class must be used! Basic types cannot be used. Use wrapper classes: `Integer`, `Float`, `Double`, `Character`, etc.

Wrapper classes are automatically used

Console

```
<terminated> ArrayListExample [Java Application]
5
10
15
3
2
[10, 15]
```

Proper printing with `System.out.println`

ArrayListExample.java

ArrayListExampleObjects.java

ArrayListExampleObjects.java | Point2D.java

```
package slides;

import java.util.ArrayList;

public class ArrayListExampleObjects {

    public static void main(String [] args) {

        ArrayList<Point2D> points;
        points = new ArrayList<Point2D>();

        points.add(new Point2D());  // Adding points on the fly
        points.add(new Point2D());

        Point2D p = new Point2D();  // Adding a point
        points.add(p);

        for(int i=0; i<points.size(); i++) {
            System.out.println("Point " + i + ": (" +
                    points.get(i).x + ", "+ points.get(i).y + ")");
        }

    }

}
```

ArrayListExampleObjects.java | Point2D.java

```
package slides;

public class Point2D {

    public double x;
    public double y;

}
```

`for` loops are commonly used to perform operations on the elements of a collection

arrays

```
int [] a = new int[10];
…
for(int i=0; i<a.length; i++) {
        int c = a[i];
        // operate with c
}
```

`ArrayList`

```
ArrayList<Point2D> points = new ArrayList<Point2D>();
…
for(int i=0; i<a.size(); i++) {
        Point2D p = points.get(i);
        // operate with p
}
```

Java offers a simplified `for` statement to deal with collections (arrays, `ArrayList`, etc.)

```
for( <element type> <element name> : <collection name>) {
    // Loop instructions
}
```

<collection name> is a valid collection reference

e.g.: a, being a declared before as `int [] a`

<element type> is the type of the elements of the collection

e.g.: `int`

<element name> is the variable which will be used inside the loop to store a copy of the current element

e.g.: c

5. Collections

Enhanced `for` for collections

ExtendedFor.java

```
package slides;

import java.util.ArrayList;

public class ExtendedFor {

    public static void main(String [] args) {

        /* array */
        double [] a = new double[10];

        for(int i=0; i<a.length; i++)       // what if we use the extended for for initialization?
            a[i] = Math.random();

        for(double c: a)
            System.out.print(c + " ");

        /* ArrayList */
        ArrayList<Point2D> points = new ArrayList<Point2D>();
        points.add(new Point2D());

        System.out.println();
        for(Point2D p: points)
            System.out.println("Point : (" + p.x + ", "+ p.y + ")" );

    }

}
```

Outline

1. Math class
2. Wrapper classes
3. String management
4. Input and output
   - Printing on the screen
   - Reading from keyboard
   - Arguments to `main`
5. Collections

> Math class

- Mathematic functions and constants
  - e, π
  - Trigonometric, logarithms, exponential, random values
- import is not required

> Wrapper classes

- Number basic datatypes have a corresponding *object-like* version
- The compiler makes automatic boxing-unboxing

> Strings

- Conversion from String > number
  - *parse* methods of wrapper classes
- Conversion from number > String
  - *valueOf* method of String
  - The compiler automatically "stringifies" numbers

Summary
Utility classes

> Printing

`System.out.println`, `System.out.printf`

> Reading

`Scanner`, `InputStreamReader + BufferedReader`

> Arguments

`args String []`

> Collections

`ArrayList`

extended `for`

Recommended lectures

The Java™ Tutorials. Oracle, **Numbers and Strings** [link]

H. M. Deitel, P. J. Deitel. *Java: How to Program. Prentice Hall, 2007 (7th Edition)*, **Chapters 29** [link]**, 30** [link]

K. Sierra, B. Bates. *Head First Java.* O'Reilly Media, 2005 (2nd Edition), **Chapter 10** [link]

H. M. Deitel, P. J. Deitel. *Java: How to Program. Prentice Hall, 2007 (7th Edition)*, **Chapter J** [link]

Programming – Grado en Ingeniería Informática	
Authors	
Of this version: Juan Gómez Romero *Based on the work by:* Ángel García Olaya Manuel Pereira González Silvia de Castro García Gustavo Fernández-Baillo Cañas	Universidad Carlos III de Madrid