Various useful classes

String manipulation
Mathematical functions
Standard input / output
File input / output
Various system features
Hashtable
Useful graphical classes

String manipulation

a string is an object instance of the class **String**

once assigned, the value of a **String** object cannot be modified
→ in practice, the modification of a string means creating a new string

String and related classes are in the java.lang package: **import java.lang.*;**

constructor:

special string constructor: **String s = "…";**

equivalent to **String s = new String("…");**

length of a string: **s.length()**

comparison:

- compare the references of the strings: **s1==s2**
  → s1 and s2 are actually the same string in memory
- compare the contents of the strings: **s1.equals(s2)**
  → s1 and s2 have the same appearance

concatenation:

special concatenation operator: **String s = s1 + s2;**

equivalent to **String s = s1.concat(s2);**

if one of the operands of + is not a string,
then it is automatically converted into a string

extraction:

index starts at 0 with first character

extraction of a character: **char c = s.charAt(index);**

extraction of a substring:

- from index_start included to the end of the string:

  String s1 = s.substring(index_start);

- from index_start included to index_end EXCLUDED:

  String s1 = s.substring(index_start , index_end);

capitalization:

convert all the characters to lower case: **String s1 = s.toLowerCase();**

convert all the characters to upper case: **String s2 = s.toUpperCase();**

conversions between data and their string representation:

string representing an object or some primitive data type value:

String s = String.valueOf(Object
or char or boolean
or int or long or float or double);

primitive data type value corresponding to a string representation **s**:

```
boolean x = Boolean.valueOf(s).booleanValue();

byte    x = Byte.valueOf(s).byteValue();

short   x = Short.valueOf(s).shortValue();

int     x = Integer.valueOf(s).intValue();

long    x = Long.valueOf(s).longValue();

float   x = Float.valueOf(s).floatValue();

double  x = Double.valueOf(s).doubleValue();
```

example:

```
String s = "abcdABCD"
System.out.println( s.substring(4) );
System.out.println( s.substring(2,6) );
System.out.println( s.toLowerCase() );
System.out.println( s.toUpperCase() );
System.out.println(s.substring(4) == "abcd".toUpperCase() );
System.out.println(s.substring(4).equals("abcd".toUpperCase()) );

String s1 = String.valueOf(123.456);
System.out.println(s1);
System.out.println( Double.valueOf(s1).doubleValue() );
```

→ ABCD
cdAB
abcdabcd
ABCDABCD
false
true
123.456
123.456

Mathematical functions

Math class contains some mathematical constants (class variables) and mathematical functions (class methods)

the **Math** class is never instantiated! it is just a sort of toolbox

Math is in the java.lang package: **import java.lang.*;**

constants:

static final double Math.PI; → 3.14 …

static final double Math.E; → 2.71 …

functions:

applied on **int**, **long**, **float** or **double** operand:

Math.abs
Math.min, Math.max

applied on **double** operand:

Math.sqrt, Math.pow, Math.exp, Math.log

Math.cos, Math.sin, Math.tan

Math.acos, Math.asin, Math.atan

Math.ceil, Math.floor, Math.round

random function: **Math.random()** returns a **double** ∈ [0.0 ; 1.0]

Standard input / output

related classes are in the java.lang and java.io packages: **import java.lang.*;**
import java.io.*;

standard streams: defined as class variables of class **System**

public static final InputStream in; = input stream

public static final PrintStream out; = output stream for printing

printing:

```
System.out.print(string);
```

System.out.println(string); = new line added at the end of string

input:

```
BufferedReader input = new BufferedReader( new InputStreamReader(System.in) );

String s;

try { s = input.readLine(); } catch (IOException e) { }
```

File input / output

related classes are in the java.io package: **import java.io.*;**

writing in file:

```
DataOutputStream output;

try { output = new DataOutputStream(new FileOutputStream("file.dat") ); }
catch (IOException e) { }


try { output.writeBoolean(b); } catch (IOException e) { }

try { output.writeByte(b); } catch (IOException e) { }

try { output.writeShort(s); } catch (IOException e) { }

try { output.writeInt(i); } catch (IOException e) { }

try { output.writeLong(l); } catch (IOException e) { }

try { output.writeFloat(f); } catch (IOException e) { }

try { output.writeDouble(d); } catch (IOException e) { }

try { output.writeUTF(string); } catch (IOException e) { }


try { output.close(); } catch (IOException e) { }
```

reading from file:

```
DataInputStream input;

try { input = new DataInputStream( new FileInputStream("file.dat") ); }
catch (IOException e) { }


try { b = input.readBoolean(); } catch (IOException e) { }

try { b = input.readByte(); } catch (IOException e) { }

try { s = input.readShort(); } catch (IOException e) { }

try { i = input.readInt(); } catch (IOException e) { }

try { l = input.readLong(); } catch (IOException e) { }

try { f = input.readFloat(); } catch (IOException e) { }

try { d = input.readDouble(); } catch (IOException e) { }

try { s = input.readUTF(); } catch (IOException e) { }


try { input.close(); } catch (IOException e) { }
```

exceptions:

if an error occurs during I/O operations
, an exception of type **IOException** is thrown

it can be in particular **FileNotFoundException** if the file was not found
or **EOFException** if the end of file has been reached

Various system features

System class is in the java.lang package: **import java.lang.*;**

time:

System.currentTimeMillis() returns **long** integer
= number of milliseconds since January 1, 1970

exit from a program:

System.exit(0);

garbage collection:

System.gc(); forces immediate and thorough garbage collection

informations about current user:

System.getProperty("user.name") returns **String**
= user name of the current user

System.getProperty("user.home") returns **String**
= home directory of the current user

System.getProperty("user.dir") returns **String**
= current working directory

File and directory access

File class is in the java.io package: **import java.io.*;**

connection between a **File** object and a **String** pathname:

```
File f = new File(path);
```
// object **f** represents a file or directory

```
String path = f.getPath();
```

tests:

f.exists() returns **true** if **f** represents a file or directory that exists

f.isDirectory() returns **true** if **f** represents a directory

f.isFile() returns **true** if **f** represents a file

list of all files and directories in **File f** :

f.list(); returns array of **String**

f.listFiles(); returns array of **File**

delete a file or directory:

```
f.delete();
```

example:

```
import java.io.*;

public class File_filedir
{
   public static void main(String[] arg)
   {
      String wd = System.getProperty("user.dir");

      System.out.println("working directory = " + wd + "\n");

      File f = new File(wd);

      File[] lf = f.listFiles();

      for (int i = 0 ; i < lf.length ; i++)
        System.out.println(  lf[i].getPath()
                        + " is file "       + lf[i].isFile()
                        + "; is directory " + lf[i].isDirectory());

      System.out.println("\nMy_dir exists " + lf[2].exists());

      lf[2].delete();

      System.out.println("\nMy_dir exists " + lf[2].exists());
   }
}
```

working directory = C:\Personal\Cmpe416\Javaprog\File_filedir

C:\Personal\Cmpe416\Javaprog\File_filedir\File_filedir.class is file true; is directory false
C:\Personal\Cmpe416\Javaprog\File_filedir\File_filedir.java is file true; is directory false
C:\Personal\Cmpe416\Javaprog\File_filedir\My_dir is file false; is directory true

My_dir exists true

My_dir exists false

Hashtable

Hashtable class is in the java.util package: **import java.util.*;**

a hashtable is an associative array:

standard array: **index → value**

generalization: **key → value**

key and **value** are objects of any kind, that is of type **Object**
(every object being an instance of either **Object** or a subclass of **Object**)

creating and filling a hashtable:

Hashtable ht = new Hashtable();

ht.size() returns number of couples (**key** , **value**) in **ht**

ht.put(key , value); enters the couple (**key** , **value**)

ht.get(key); returns the value associated to **key**

ht.remove(key); returns the couple (**key** , associated value)

ht.clear(); removes all the couples (**key** , **value**)

tests on hashtable:

ht.contains(value) returns **true** if and only if **ht** contains couple (-,**value**)

ht.containsKey(key) returns **true** if and only if **ht** contains couple (**key**,-)

ht.isEmpty() returns **true** if and only if **ht** is empty

miscellaneous:

Hashtable ht1 = ht.clone(); makes a copy of the hashtable

ht.toString() returns a string representation of **ht**

example:

```
Hashtable ht = new Hashtable();

ht.put("January"  , "31_days");
ht.put("February" , "28_days");
ht.put("March"    , "31_days");
ht.put("April"    , "30_days");

System.out.println("number of elements of hashtable = " + ht.size());

System.out.println("February has " + ht.get("February"));
System.out.println("April has " + ht.get("April"));

System.out.println("February key present in hashtable? "
                   + ht.containsKey("February"));

System.out.println("32_days value present in hashtable? "
                   + ht.contains("32_days"));
```

→ number of elements of hashtable = 4
February has 28_days
April has 30_days
February key present in hashtable? true
32_days value present in hashtable? false