About Java

Sun Microsystems. Public announcment 1995. Platform-independent.
The compiler creates from source code files (*.java) the bytecode files (*.class).
The Java Virtual Machine(JVM) translates the bytecode into code that can run on the concrete computer. Thus, a Java program can run on any platform for which the JVM is implemented and on any computer on which the JVM is installed.

To run a Java program:
Java Runtime Environment (JRE) in folder "Program Files\Java\jre1.8.x"
http://www.oracle.com/technetwork/java/javase/downloads/index.html

To develop Java software:

1. Java Development Kit (JDK) in folder "Program Files\Java\jdk1.8.x"
2. Integrated Development Environment (IDE):
   - NetBeans
   - Eclipse (**http://www.eclipse.org**)
3. Help **http://docs.oracle.com/javase/7/docs/api/**

Java SE – standard edition, Java EE –enterprise edition

Java programs: applications, applets, servlets.

Main concepts (1)

File FirstJava.java (the name of file and the name of class must match):

```
package firstjava; // The package has the same name as the project. All the classes
// implemented in a project must be the members of the project package. Generally,
// a package is hierarchical: it may contain classes and other packages.
// In NetBeans the package names are in lowercase letters (the project name was FirstJava)
public class FirstJava {  // The class has also its access modifier
// Usually, the class containing main() has the same name as the project
    public static void main(String[] args) {
        // args is an array of strings retrieved from command line
        // The main() function returns nothing (void).
        // Naturally, the main() must be a public method
        // And of course, main() must be static because it is called at the beginning
        // when there are no objects from class FirstJava.
    }
} // no semicolon at the end
```

Differences from C++: no preprocessor, no *.h files, no functions out of classes, no access control labels, packages instead of namespaces, each class must be completely defined in its own *.java file.

Main concepts (2)

Folders we need: the working folder and its subfolder called as the package folder. In our example let C:\JavaExamples be the working folder. The name of the package folder must match the name of package (in our example C:\JavaExamples\firstjava). All the *.java files of a package must be together in the package folder. The working folder should be also the current folder.

The Java compiler is "Program Files\Java\jdk1.8.x\bin\javac.exe". Examples:

1. Compile file FirstJava.java:

```
C:\JavaExamples>"C:\Program Files\Java\jdk1.8.0_111\bin\javac" firstjava\FirstJava.java
```

2. Compile all the files of package firstjava:

```
C:\JavaExamples>"C:\Program Files\Java\jdk1.8.0_111\bin\javac" firstjava\*.java
```

The results are *.class files located in the package folder. Linking is not needed.

To run Java program, start the JVM which is implemented in "Program Files\Java\jdk1.8.x\bin\java.exe" and also in "Program Files\Java\jre1.8.x\bin\java.exe". If the package folder is the subfolder of the current folder, the java.exe command line parameter is the complete name (*PackageName.ClassName*) of class containing the main() function. Example:

```
C:\JavaExamples>"Program Files\Java\jre1.8.0_111\bin\java" firstjava.FirstJava
```

If this is not the case, the classpath must be specified.

Main concepts (3)

The classpath sets the path from the current folder to the folder containing the package folder. The classpath does not contain the package folder itself. To specify the classpath, set the CLASSPATH Windows environmental variable or use the java.exe –classpath command line parameter. Example:

```
C:\Documents and Settings\viktor>"c:\Program Files\Java\jre1.8.0_111\bin\java"
 –classpath "..\..\JavaExamples" firstjava.FirstJava
```

Java archive *.jar contains all the *.class files and the manifest.
Example of class Manifest.txt from the C:\JavaExamples working folder.
Main-Class: firstjava.FirstJava
Class-Path: firstjava.jar
The last line must end with newline (i.e. at the end press ENTER). The archive is created with "Program Files\Java\jdk1.8.x\bin\jar.exe". The working folder should be also the current folder. Example:

```
C:\JavaExamples>"c:\Program Files\Java\jdk1.8.0_111\bin\jar" cfmv FirstJava.jar
Manifest.txt firstjava\*.class
```

c – create new archive; v – verbose; fm – the second parameter is the archive name, the third parameter is the manifest name.
The FirstJava.jar will be stored in the working folder.

Main concepts (4)

To run an application packed into archive use the –jar command line parameter. Example:

```
C:\Documents and Settings\viktor>"c:\Program Files\Java\jre1.8_0111\bin\java" –jar
"c:\JavaExamples\FirstJava.jar"
```

In addition to *.class files, an archive may contain also figures (*.gif, *.png. *.jpg), web pages (*.html) and other files used by the current application.

To open the archive use WinZip.

Java data types

byte – 8-bit signed integer from –128 to +127.
short – 16-bit signed integer
int – 32-bit signed integer
long – 64-bit signed integer
There are no unsigned integers.
float – 32-bit floating point number
double – 64-bit floating point number
char – 16-bit Unicode character
boolean – 8-bit logical value true or false; result of relational operations like == or !=, required in logical operations and conditional expressions in statements such as if, for and while.

```
int i = 100;   double d = 3.14;   char c = 'A';   boolean b = true;
i = d;  // error, explicit cast i = (int)d; is required
d = i;  // correct, implicit cast is allowed
b = i;  b = (boolean)i; // errors, impossible cast
i = b;  i = (int)b; // errors, impossible cast
c = i; // error, explicit cast c = (char)i; is required
i = c;  // correct, implicit cast is allowed
```

About Java operators

```
int i1, i2, i3;
if (i1) {….} // error, correct is if (i1 != 0) {…}
i1 = i2 && i3; // error, the logical operations need boolean operands,
                     // the result is also boolean
boolean b1, b2, b3;
b3 = b1 + b2; // error, arithmetics with booleans is not defined

// Let fun() to be a function returning boolean
b1 = b2 && fun(); // if b2 == false, fun() is not called
b1 = b2 || fun(); // if b2 == true, fun() is not called
b1 = b2 & fun(); // in any case fun() is called (short-circuit logical operator)
b1 = b2 | fun(); // in any case fun() is called

i1 = i2 & i3; // bitwise AND
i1 = i2 | i3; // bitwise OR
```

Classes and objects (1)

```
package geometry;
public class Triangle { // Each class has its own access modifier (private or public)
        private int m_Side1 = 0;
        private int m_Side2 = 0;
        private int m_Side3 = 0;
        // Each attribute (field) has its own access modifier (private, protected or public)
        // Optionally, the attributes may also have initial values which stay valid until
        // a constructor or some function has not changed them
        public Triangle(int s1, int s2, int s3) { // Constructor with arguments. As any
        // other function, the constructor has also its own access modifier.
             m_Side1 = s1;  m_Side2 = s2;  m_Side3 = s3;
        }
        public Triangle() { } // Empty constructor.
        public int GetPerimeter() {
           return m_Side1 + m_Side2 + m_Side3;
        }
}
// If the access modifier is missing, the class, field or method is public in its package and
// private outside the package.
```

Classes and objects (2)

All the objects must be created dynamically (with the new operator).

Triangle t1 = new Triangle(5, 7, 12); // allocate memory and call constructor

By Java terminology, t1 is the reference to object of type Triangle. Actually, we can think about references as pointers (and not as C++ references). But in Java there are neither pointer arithmetic nor the * and & operators.

t1.GetPerimeter(); // Java has no the -> operator

Triangle t2 = new Triangle(); // allocate memory and call (here the empty) constructor

The parenthesis are obligatory (in C++ the call to constructor without arguments does not need parenthesis).

Classes without constructors are allowed. In that case the system itself generates an empty constructor. If there is at least one user-defined constructor, the empty constructor will be not generated and if it is needed, the programmer must write it himself.

Triangle t3; // t3 has value null – we have an uninitialized pointer only.

t3.GetPrimeter(); // run-time error, triangle t3 does not exist

In C++ *Triangle t3* defines a local or global object and calls the constructor not having arguments.

Deallocation is not needed, the garbage collector does it automatically. Destructors are unknown in Java. If necessary, add to class protected void finalize() {…… }. This function is automatically called prior to garbage collection.

Classes and objects (3)

```
package geometry;  // the current package
import java.awt.*;  // as using namespace in C++, to avoid writing complete class names
public class Triangle {
    private Point m_A = new Point(0, 0); // the complete name is java.awt.Point
    private Point m_B = new Point(0, 0);
    private Point m_C = new Point(0, 0);
    public Triangle(int xa, int ya, int xb, int yb, int xc, int yc) {
        m_A.setLocation(xa, ya);
        m_B.setLocation(xb, yb);
        m_C.setLocation(xc, yc);
    }
    public Triangle() { }
    public double GetSideAB() { return m_A.distance(m_B);  }
    public double GetSideBC() {  return m_B.distance(m_C);  }
    public double GetSideAC() {  return m_A.distance(m_C);  }
    public double GetAngleA() {
            double ab = GetSideAB(), bc = GetSideBC(), ac = GetSideAC();
            return Math.acos((ac * ac+ ab * ab - bc * bc) / (2 * ac * ab));
    } // acos is the static method of class java.lang.Math
}
```

Arrays (1)

In C++ we have 4 modes to declare an array:

```
int mi[5];
int *pmi = new int[5];
vector<int> iv(5);  // object
vector<int> *piv = new vector<int>(5); // pointer to object
```

In Java an array is always object:

```
int mi[] = new int[5]; // mi is the reference (pointer) to array consisting of 5 integers
int mi[]; // mi has value null. The array has no memory
int mi[] = { 1, 2, 3, 4, 5 }; // declaring and initializing

Triangle mt[] = new Triangle[2];  // mt consists of 2 references, their value is null
                                  // Triangles do not exist yet
mt[0] = new Triangle(); // the first index is always 0
mt[1] = new Triangle(0, 0, 5, 5, 10, 10);
Triangle mt[] = { new Triangle(), new Triangle(0, 0, 5, 5, 10, 10) };
                               // declaring and initializing
```

Arrays (2)

Each array has public attribute length:

```
for (int i = 0;  i < mi.length;  i++)
      mi[i] = i + 1;

int mmi[][] = new int[2][3];  // matrix of 2 rows and 3 columns
int mmi[][] = { { 1, 2, 3} , {10, 20, 30} };
```

You may also write

```
int[] mi = new int[5];
int[][] mmi = new int[2][3];
```

Class java.util.Arrays provides several useful static methods for array operations.
Examples:

```
Arrays.sort(mi);
int j = Arrays.binarySearch(mi, i);
// Returns the index of member equal to i. If not found, returns -1
Arrays.fill(mi, i, j, k); // Sets mi[i]….mi[j] to k
int mi_section = Arrays.copyOfRange(mi, i, j);
// Creates new array consisting of mi[i]….mi[j]
```

Strings

In Java strings are always objects of class String.

```
String s; // empty pointer, has value null
String s1 = "Hello"; // s1 points to string containing text “Hello”
String s2 = ""; // s2 points to string not containing characters
```

Objects of class String are immutable: it is not possible to alter the text in string.
To change the text, you have to create a new string that contains the modified text.
Examples of constructors and methods:

```
String s3 = "world";
String s4 = s1 + " "+ s3;  // " " is the string constant
if (s1.equals(s2)) {……………….}
if (s1.compareTo(s2) < 0  {……………….}
if (s2.isEmpty()) {……………….}
int i = s4.length();
int j = s4.indexOf('e'); // -1 if character ‘e’ not found
int k = s4.indexOf(s3); // -1 if text “world” not found
char c = s4.charAt(3); // if the index (starts from 0) is not correct, throws exception
String s5 = s4.replace(' ', '+');
String s6 = s5.toUpper();
String s7 = new String(s6); // copying constructor
```

Argument passing

C - call by value
C++ - call by value and call by reference
Java - only call by value

```
void fun(int iArg, Triangle tArg) {
……………………………………..
}

int i = 10;
Triangle t = new Triangle();
fun(i, t);
```

Value of i is copied into iArg. Changes of iArg do not affect the value of i.
Pointer t is copied into tArg. Function fun() can change the attributes of triangle.
It means that actually we have the call by reference.

Java standard output

Question - what function replaces printf?
Standard class System has member:

```
static PrintStream out; // static - we do not need objects of class System
```

Standard class PrintStream has methods like:

```
void println(String s);
void println(int i);
void println (boolean b);
void println(char c);
```

……………………………..

They all print the value of argument into the console window. Consequently:

```
System.out.println("Hello world"); // prints "Hello world"
System.out.println(100); // prints 100
int x = 200;
System.out.println(x); // prints 200
boolean b = false;
System.out.println(b); // prints false
```

Final variables

The final keyword in Java and the const keyword in C++ are a bit different.

```
public class ccc {
        final int m_Var = 100;
        final int m_Arr[] = {100, 200, 300};
        …………………………………………
        public void fun() {
                m_Var = 200; // Error, m_Var is a constant
                m_Arr[0] = 1000; // Correct, a final pointer does not obstruct to change
                                                  // the members of a object
                m_Arr = new int[10]; // Error, you cannot rebind a final pointer to another object
                ………………………………
        }
}
```

Exception handling (1)

In Java the exception thrown out from a function is always an object.

1. Unchecked exceptions:

```
String s = "Hello";
int i;
char c = s.charAt(i);
```

Here the usage of try-catch is not obligatory.

If i < 5, everything is OK. If not, charAt() throws an object of Java standard class StringIndexOutOfBoundsException.

As the try-catch is missing, the exception moves on to higher level and so on. If even in the main() function the corresponding try-catch was not found, the program terminates.

If you are not sure that i will have always correct values, write

```
char c;
try {
    c = s.charAt(i);
}
catch (StringIndexOutOfBoundsException ex) {
     System.out.println(ex.getMessage()); // each exception has the getMessage() method
}
```

Exception handling (2)

2. Checked exceptions:
Call to function throwing checked exceptions must be wrapped into try-catch. If not, you cannot compile your code. Example: IOException thrown by functions handling files.

The function must list the exceptions it may throw:

```
public void fun(int i) throws IllegalArgumentException {
     if (i < 0)
         throw new IllegalArgumentException("Argument is negative");
 ……………………………………………………………………
}

try {
    fun(-1);
}
catch (IllegalArgumentException ex) {
    System.out.println(ex.getMessage());
    return;
}
```

Exception handling (3)

Java library defines about 30….40 exception classes, they are all derived from base class Exception.

```
public void fun(int i) throws IllegalArgumentException,
                              IndexOutOfBoundsException,
                              NullPointerException  {
…………………………………………………………………………..
 }

try {
    fun(-1);
}
catch (IllegalArgumentException argex) {
    System.out.println(argex.getMessage()); // getMessage() is inherited from Exception
    return;
}
catch (Exception ex) { // all the other exceptions, must be the last catch
    System.out.println(ex.getMessage());
    return;
}
```

Type wrappers and autoboxing (1)

Two different types of variables:

```
int i; // 4-byte integer
String s; // reference, i.e. pointer
```

Type wrapper is a class encapsulating a primitive type (like int) within an object. The Java type wrapper classes are Double, Float, Integer, Long, Short, Byte, Boolean, Character.

If you need to pack an integer into object, write

```
Integer iObj = new Integer(i);
```

To unpack, write

```
i = iObj.intValue();
```

The wrapper classes are immutable - you cannot change the wrapped value.

Parsing:

```
String s;
int i;
try {
      i = Integer.parseInt(s); // parseInt() is static
}
catch (NumberFormatException ex) { // obligatory, checked exception
      System.out.println(s + " is not a correct integer");
}
```

Type wrappers and autoboxing (2)

Converting to string:

```
int i = 100;
String s = Integer.toString(i); // toString() is static
```

Autoboxing: you may write

```
Integer iObj = i; // the same as Integer iObj = new Integer(i);
i = iObj; // the same as i = iObj.intValue();
```

Java performs autoboxing whenever a conversion from primitive type to wrapper class object and vice versa is required. Examples:

1. `public void fun(Integer iObj) {…………………….}`

If you call it

```
fun(100);
```

autoboxing takes place

2. `public void fun(int i) {…………………….}`

If you call it

```
Integer iObj = new Integer(100);
fun(iObj);
```

autoboxing takes place

Type wrappers and autoboxing (3)

3. System.out.println("The result is " + i.toString());

Error - here the autoboxing for variable i is not performed. The correct is expression is

System.out.println("The result is " + i);

The String operator+ function converts the additives of primitive types to strings.

4. Integer x = 100, y = 200, z; // primitive types are abandoned

z = x + y;

Due to autoboxing it works but is very unefficient.

File FirstJava.java

```java
package firstjava;
public class FirstJava {
    public static void main(String[] args) {
        if (args.length != 6) {
            System.out.println("Triangle is not correctly specified");
        }
        int Coord[] = new int[6];
        for (int i = 0;  i < 6;  i++) {
            try {
                Coord[i] = Integer.parseInt(args[i]);
            }
            catch (NumberFormatException nex) {
                System.out.println(args[i] +  " is not a correct integer");
                return;
            }
        }
        Triangle t = new Triangle(Coord);
        try {
            System.out.println("Angle A is " + t.GetAngleA() +  " (in degrees)");
            System.out.println("Angle B is " + t.GetAngleB() +  " (in degrees)");
            System.out.println("Angle C is " + t.GetAngleC() +  " (in degrees)");
        }
        catch (ArithmeticException aex) {
            System.out.println(aex.getMessage());
        }
        System.out.println("Side AB is " + t.GetSideAB());
        System.out.println("Side AC is " + t.GetSideAC());
        System.out.println("Side BC is " + t.GetSideBC());
        System.out.println("Perimeter is " + t.GetPerimeter());
        System.out.println("Area is " + t.GetArea());
        return;
    }
```

File Triangle.java

```
package firstjava;
import java.awt.*;
public class Triangle {
    private Point m_A = new Point(0, 0);
    private Point m_B = new Point(0, 0);
    private Point m_C = new Point(0, 0);
    public Triangle(int xa, int ya, int xb, int yb, int xc, int yc) {
        m_A.setLocation(xa, ya);
        m_B.setLocation(xb, yb);
        m_C.setLocation(xc, yc);
    }
    public Triangle(int coord[]) throws IllegalArgumentException {
        if (coord.length != 6)
            throw new IllegalArgumentException();
        m_A.setLocation(coord[0], coord[1]);
        m_B.setLocation(coord[2], coord[3]);
        m_C.setLocation(coord[4], coord[5]);
    }
    public Triangle() { }
    public boolean equals(Triangle t) {
        if (m_A == t.m_A && m_B == t.m_B && m_C == t.m_C)
            return true;
        else
            return false;
    }
    public String toString() {
         return "A = " + m_A + " B = " + m_B + " C = " + m_C;
    }
    protected Triangle clone() {
         return new Triangle(m_A.x, m_A.y, m_B.x, m_B.y, m_C.x, m_C.y);
    }
```

```
public double GetSideAB() {
    return m_A.distance(m_B);
}
public double GetSideBC() {
    return m_B.distance(m_C);
}
public double GetSideAC() {
    return m_A.distance(m_C);
}
public double GetPerimeter() {
    return GetSideAB() + GetSideBC() + GetSideAC();
}
double GetArea() {
     double SignedArea = 0.5 * (m_A.getX() * (m_B.getY() - m_C.getY())
            +  m_B.getX() * (m_C.getY() - m_A.getY())
            +  m_C.getX() * (m_A.getY() - m_B.getY()));
     return Math.abs(SignedArea);
 }
public double GetAngleA() throws ArithmeticException {
     double ab = GetSideAB(), bc = GetSideBC(), ac = GetSideAC();
     if (ac == 0)
         throw new ArithmeticException("The length of side AC is 0");
```

```
            if (ab == 0)
                throw new ArithmeticException("The length of side AB is 0");
           return Math.acos((ac * ac+ ab * ab - bc * bc) / (2 * ac * ab)) * 180 / Math.PI;
        }
        public double GetAngleB() throws ArithmeticException {
            double ab = GetSideAB(), bc = GetSideBC(), ac = GetSideAC();
            if (bc == 0)
                throw new ArithmeticException("The length of side BC is 0");
            if (ab == 0)
                throw new ArithmeticException("The length of side AB is 0");
            return Math.acos((bc * bc + ab * ab - ac * ac) / (2 * bc * ab)) * 180 / Math.PI;
        }
        public double GetAngleC() throws ArithmeticException {
            double ab = GetSideAB(), bc = GetSideBC(), ac = GetSideAC();
            if (bc == 0)
               throw new ArithmeticException("The length of side BC is 0");
            if (ac == 0)
               throw new ArithmeticException("The length of side AC is 0");
            return Math.acos((bc * bc + ac * ac - ab * ab) / (2 * bc * ac)) * 180 / Math.PI;
        }
    }
```

Inheritance (1)

public class *subclass_name* extends *superclass_name* {…………………..}

```
public class Item {
    protected long m_Code = 0;
    protected String m_Name = "";
    public Item(long c, String n) { m_Code = c;  m_Name = n; }
}

public class StockItem extends Item {
    protected int m_Quantity = 0;
    protected double m_Price = 0;
    public StockItem(long l, String s, int q, double p) {
        super(l, s);  // Use "super" instead of base class name
                      // Must be the first expression
                      // If the base class constructor does not have arguments,
                      // super(); is omitted (but may be written).
        m_Quantity = q;
        m_Price = p;
    }
}
```

Inheritance (2)

```
public class Item {
        ……………………………………………
        public String toString() {
                return "Item " + m_Name + " with code " + m_Code;
        }
}

public class StockItem extends Item {
        ……………………………………………..
        public String toString() {
                return super.toString() + ", " + m_Quantity + " units in stock, price " +
                                        m_Price;
        }
// StockItem.toString() overrides Item.toString(). To use the overrided base class
// members, write super.member_name.
}
```

Overriding - two functions have the same name, return value and parameters (their signatures match).

Overloading - two functions have the same name, but there are differences in return values and/or parameters (example - constructors)

Inheritance (3)

All the base class methods except the private ones are virtual by default. Calls to overridden methods are resolved at run time, not in the course of compiling. In Java it is referred as dynamic method dispatch (late binding in C++). Let us have

```
public class Shoe extends StockItem {……………………}
public class TShirt extends StockItem {……………………}
```

etc., they all have their own method

```
public String toString() {……………………}
```

We may write:

```
StockItem item1 = new Shoe(…..), item2 = new TShirt(……);
```

Then due to polymorhism

```
System.out.println(item1.toString());  // prints the description of a shoe
System.out.println(item2.toString());  // prints the description of a T-shirt
```

Inheritance (4)

Let us have also

```
public class StockItem extends Item {
      ……………………………………….
      public void SetSalePrice() { m_Price *= SaleCoeff(); }
      protected double SaleCoeff() {  return 1; }
}
```

Classes Shoe, TShirt, etc. have their own SaleCoeff() method. Then in

```
TShirt shirt1 = new TShirt(….);
shirt1.SetSalePrice();
```

the SetSalePrice() defined in StockItem uses the SaleCoeff() from TShirt.

```
abstract public class StockItem extends Item {// abstract class contains abstract methods
// It is not possible to create objects from abstract classes
      ……………………………………….
      public void SetSalePrice() { m_Price *= SaleCoeff(); }
      abstract protected double SaleCoeff(); // abstract method has no body
}
```

Inheritance (5)

Interface may be considered as an abstract class in which all the methods are abstract and all the attributes (if any) are static and final.

```
public interface IntegerStack {
      void Push(int i); // keyword abstract is not written
      int Pop();
}

public MyStack implements IntegerStack { // implements, not extends
      private int m_Capacity;
      public MyStack(int n) {……}
            // All the methods of implementing class must be public
            // Absolutely all the methods specified in the interface must be implemeted
            // Additional methods and attributes can be added
      public void Push(int i) {……}
      public int Pop() {……}
}
```

A class may implement several interfaces. A class may extend its base class and also implement interface(s):

```
public class ccc extends bbb implements iii1, iii2 {……………….}
```

Inheritance (6)

```
final void public fun() {……………}
```

Now overriding of this function is not possible

```
final public class ccc {…………….}
```

Now deriving of other classes from this class in not possible

Base class Object (1)

All the classes (standard and created by us) are derived from base class Object. Therefore each class inherits 11 standard methods. Some of them are marked as final - they are ready to work and cannot be overridden. The others are empty (do nothing) and if the user wants, he may override them. Some examples:

1. getClass().getName() returns the string with class full name. Method getClass() is final.

```
Triangle t;
System.out.println(t.getClass().getName()); // Prints "firstjava.Triangle"
```

2. String toString(); // If you need it, override

```
 public String toString() {
        return "A = " + m_A + " B = " + m_B + " C = " + m_C;
 }
```

Example result:

```
A = java.awt.Point[x=1,y=4] B = java.awt.Point[x=6,y=9] C = java.awt.Point[x=12,y=45]
```

Usage:

```
System.out.println("Triangle vertexes: " + t.toString());
System.out.println("Triangle vertexes: " + t);
System.out.println(t);
```

Base class Object (2)

3. boolean equals(Object object); // If you need it, override

```
 public boolean equals(Triangle t) {
        if (m_A == t.m_A && m_B == t.m_B && m_C == t.m_C)
            return true;
        else
            return false;
 }
```

4. protected Object clone(); // If you need it, override

```
 protected Triangle clone() {
         return new Triangle(m_A.x, m_A.y, m_B.x, m_B.y, m_C.x, m_C.y);
}
```

5. void finalize(); // Destructor. If you need it, override

Base class Object (3)

In C++:

```
void fun(void *p) {……………..}
```

p may point to any variable.

In Java:

```
void fun(Object ob) {……………..}
```

As all the classes are derived from Object, ob may be a reference to any object.

```
Object mOb[] = new Object[10];
```

Array mOb may contain references to objects of different classes.

Enumerations

Enumeration is a list of named constants.

```
public enum CreditCard {
        MasterCard, VISA, AmericanExpress, DinersClub
} // enum may be a member of a package or an internal member of a class
In some other class:
CreditCard m_CreditCard; // attribute of type CreditCard
……………………………….
m_CreditCard = CreditCard.VISA; // new() is not needed
……………………………….
if (m_CreditCard == CreditCard.MasterCard) {
………………………………..
}
switch (m_CreditCard) {
        case MasterCard: // do not write CreditCard.MasterCard
                ……………………………………………..
        case VISA:
                ……………………………………………..
        ……………………………………………….
}
System.out.println("The customer has " + m_Card);
```

Input/output (1)

```
import java.io.*;

File DataFile = new File("c:\\temp\\data.bin");
```

The string argument of this constructor may specify an existing or non-existing file or folder. Therefore we have to check what we have got:

```
 if (DataFile.exists()) {
      if (!DataFile.isFile()) {…………..}
     if (!DataFile.canWrite() || !DataFile.canRead()) {…………..}
}
else {
       try {
            DataFile.createNewFile();
       }
       catch (IOException ioe) {……………. } // checked exception
}
```

Input/output (2)

To read and write, create input and output streams and bind them to File object. Usually those operations are carried out in the following way:

1. In case of binary data

```
FileOutputStream Output;
BufferedOutputStream BufOutput;
 try {
     Output = new FileOutputStream(DataFile, false);
// false - destroy the current contents, start writing from the beginning
// true - append new data to the end of file
     BufOutput = new BufferedOutputStream(Output, 1024);
// 1024 - the length of buffer
}
catch (IOException ex1) {……………}
FileInputStream Input;
BufferedInputStream BufInput;
 try {
     Input = new FileInputStream(DataFile);
     BufIntput = new BufferedInputStream(Input, 1024);
}
catch (IOException ex2) {……………}
```

Input/output (3)

2. In case of Unicode texts:

```
FileWriter Writer;
BufferedWriter BufWriter;
 try {
        Writer = new FileWriter(DataFile, false);
// false - destroy the current contents, start writing from the beginning
// true - append new data to the end of file
        BufWriter = new BufferedWriter(Writer, 1024);
// 1024 - the length of buffer
}
catch (IOException ex1) {……………}
FileReader Reader;
BufferedReader BufReader;
 try {
        Reader = new FileReader (DataFile);
        BufReader = new BufferedReader(Reader, 1024);
}
catch (IOException ex2) {……………}
```

Input/output (4)

Binary streams: reading and writing

```
byte Buf[] = new byte[n];
try {
      BufOutput.write(Buf, 0, n);  // write n bytes starting from Buf[0]
}
catch (IOException ex) {……………...}
```

Before closure:

```
try {
      BufOutput.flush();
}
catch (IOException ex) {……………...}
int m;
try {
      m = BufInput.read(Buf, 0, n);  // read n bytes, store them starting from Buf[0]
}
catch (IOException ex) {……………...}
if (m == -1) {…………..} // end of file reached, nothing read
else if (m < n) {…………..} // no enough data
```

Input/output (5)

Text streams: reading and writing

```
char Buf[] = new char[n];
// to convert string s to array of characters, write char Buf = s.toCharArray();
try {
      BufWriter.write(Buf, 0, n);  // write n characters starting from Buf[0]
}
catch (IOException ex) {……………..}
```

Before closure:

```
try {
      BufWriter.flush();
}
catch (IOException ex) {……………..}
int m;
try {
      m = BufReader.read(Buf, 0, n);  // read n characters, store them starting from Buf[0]
}
catch (IOException ex) {……………..}
if (m == -1) {…………..} // end of file reached, nothing read
else if (m < n) {…………..} // no enough data
String s = String(Buf, 0, m); // to convert the got data to string
```

Input/output (6)

Text streams: reading and writing by lines

```
String s;
try {
      BufWriter.write(s + \'n');  // write the contents of string, append the end of line
}
catch (IOException ex) {……………..}
```

Before closure:

```
try {
      BufWriter.flush();
}
catch (IOException ex) {……………..}

String s;
try {
      s = BufReader.readLine();  // the end of line ('\n', '\r' or "\r\n") at the end is not read
}
catch (IOException ex) {……………..}
if (s == null) {…………..} // end of file reached, nothing read
```

Input/output (7)

Reading from keyboard:

```
System.out.println("Type an integer");
BufferedReader KeyboardReader;
KeyboardReader = new BufferedReader(new InputStreamReader(System.in));
String s = null;
try {
    s = KeyboardReader.readLine();
}
catch (IOException ex) {……………….}
if (s != null) {
    int i;
    try {
        i= Integer.parseInt(s);
    }
    catch (NumberFormatException ex) {……………….}
}
```

Closure: all the streams have method close(), for example

```
try {
    BufIntput.close();
}
catch (IOException ex) {……………}
```

Threads (1)

How to create thread:

1. Create public class MyThread extends Thread {…………………..}
2. In MyThread override public void run() {……………..}
(actually the main() for the thread).
3. Create MyThread m_Thread = new MyThread(……..);
4. Call m_Thread.start(); or put the call to start() into the MyThread constructor.

It may be useful to add into MyThread constructor
setDaemon(true); // before call to start()
A daemon thread stops automatically when the thread (including main()) that started it stops.

Critical sections:

1. Select a synchronization object. It can be any object. For example, if you want to control operations with an array, this array may be the object. Or even better, declare
final Object m_Lock = new Object();
In C++ the synchronization object was a structure of type CRITICAL_SECTION. Of course, the threads you want to synchronize must be able to access the synchronization object.
2. synchronized (m_Lock) { // in C++ EnterCriticalSection()
……………………………… // synchronized operations
} // in C++ LeaveCriticalSection()

Threads (2)

<u>Synchronized methods</u>:

```
public synchronized void fun(…) {………….}
```

means that while a thread is inside a synchronized method, all the other threads trying to call this method must wait. In other words, the complete method is a critical section. Synchronization object is not needed. Moreover, if a class has more than one synchronized method and one of them is being called by a thread, the other threads are not allowed to call any other synchronized method too.

<u>Interthread communication</u>:

1. Select an event object. It can be any object, but the best solution is to declare a dedicated lock:

```
final Object m_Lock = new Object();
```

In C++ the event object was specified by handle created by CreateEvent(). Here we can think that m_Lock is as an auto-reset event and its initial state is nonsignaled.The threads you want to communicate with each other must be able to access the event object.

For interthread communication, class Object (and consequently any other class) has the following functions:

wait() and wait(long timeout_in_ms) correspond to WaitForSingleObject().

notify() and notifyAll() correspond to SetEvent().

As Java events are auto-reset ones, ResetEvent() is not needed.

Threads (3)

2. To force a thread to wait, write

```
synchronized (m_Lock) { // must be in synchronized statement or method
    try {
        m_Lock.wait(); // m_Lock.wait(1000) -waits max 1 second
    }
    catch (InterruptedException ex) {
        return; // thrown when a thread which is waiting or otherwise occupied
                // is interrupted, i.e. the waiting ends in abnormal way
    }
}
```

3. To wake up the waiting thread, write

```
synchronized (m_Lock) { // must be in synchronized statement or method
    m_Lock.notify();
}
```

If several threads are waiting the signal, use notifyAll().

4. Problem: if one of the threads sends the notification too early (i.e. when the other thread is not in the waiting state yet), this notification "flies into the space". As the result, when the other thread at last fells asleep, it may be never wake up again. To confirm that you actually need to wait before calling the wait() method, use boolean flags.

Threads (4)

Example:

```
boolean m_MustWait = true;
synchronized (m_Lock) {
    while (m_MustWait) {
        try {
            m_Lock.wait();
        }
        catch (InterruptedException ex) {
            return;
        }
     }
     m_MustWait = true;
}

synchronized (m_Lock) {
    m_MustWait = false; // if not waiting yet, will not start to wait
    m_Lock.notify(); // if waiting, will stop it
}
```

Threads (5)

Ending threads:

Do not use deprecated methods like suspend(), resume() and stop(). A thread ends when the run() method returns.

```
MyThread m_Thread = new MyThread(……..);
…………………………………………………….
try {
     MyThread.join(); // waits for the thread to finish
                      // MyThread.join(1000) -waits max 1 second
catch (InterruptedException ex {
…………………………..
}
```

Ending application:

```
System.exit(return_code); // static, argument is an integer
// Actually terminates the JVM
```

Suspending a thread for a while:

```
Thread.sleep(time_to_sleep_in_ms); // static, argument is a long integer
```

File JavaThreads.java

```
package javathreads;
import java.io.*;
public class JavaThreads {
    public static void main(String[] args) {
        Counter count = new Counter();
        Boolean stop = false;
        Producer pro = new Producer(count, stop);
        Consumer con = new Consumer(count, stop);
        System.out.print("Press ENTER to stop"); // "Press any key" is difficult to implement
        try {
            System.in.read();
        }
        catch (IOException ex) {  }
        stop = true; // autoboxing
        try {
            pro.join();
            con.join();
        }
        catch (InterruptedException ex) { }
        System.exit(0);
    }
}
```

File Counter.java

```
package javathreads;
public class Counter {
    private int m_Counter = 0;
    private boolean m_MustWait = false;
    final private Object m_Lock = new Object();
    public void Increment() {
       synchronized (m_Lock) {
           while (m_MustWait) {
                try {
                   m_Lock.wait();
                }
                catch (InterruptedException ex) {
                    return;
                }
            }
            m_Counter++;
            m_MustWait = true;
            System.out.println("Incrementing: counter value is " + m_Counter);
            m_Lock.notify();
       }
```

```
        }
        public void Decrement() {
            synchronized (m_Lock) {
                while (!m_MustWait) {
                    try {
                        m_Lock.wait();
                    }
                    catch (InterruptedException ex) {
                        return;
                    }
                }
                m_Counter--;
                m_MustWait = false;
                System.out.println("Decrementing: counter value is " + m_Counter);
                m_Lock.notify();
            }
        }
    }
```

File Producer.java

```
package javathreads;
public class Producer extends Thread {
    volatile private Counter m_Counter;
    volatile private Boolean m_Stop;
    public Producer(Counter c, Boolean stop) {
        m_Counter = c;
        m_Stop = stop;
        start();
    }
    public void run() {
        System.out.println("Producer started");
        while (!m_Stop) // autoboxing
            m_Counter.Increment();
     }
}
```

File Consumer.java

```
package javathreads;
public class Consumer extends Thread {
    volatile private Counter m_Counter;
    volatile private Boolean m_Stop;
    public Consumer(Counter c, Boolean stop) {
        m_Counter = c;
        m_Stop = stop;
        start();
    }
    public void run() {
        System.out.println("Consumer started");
        while (!m_Stop) // autoboxing
            m_Counter.Decrement();
    }
}
```

Sockets (1)

```
import java.net.*;
import java.io.*;
private InetAddress m_Address; // five objects we need
private Socket m_Socket;
private int m_Port = 1234;
private InputStream m_InStream; // FileInputStream etc. are derived from InputStream
private OutputStream m_OutStream;
 try { // static factory methods instead of constructor
        m_Address = InetAddress.getByAddress(new byte[] {127,0,0,1});
}
catch (UnknownHostException ex1) {…………..}
try {// creates and connects client socket
        m_Socket = new Socket(m_Address, m_Port);
}
catch (IOException ex2) {…………..}
try {// binds with data streams
        m_InStream = m_Socket.getInputStream();
        m_OutStream = m_Socket.getOutputStream();
}
catch (IOException ioe) {………….}
```

Sockets (2)

```
int nRead, Offset, nToRead, nToWrite;
byte Buf[] = new byte[2048];
try {
    nRead = m_InStream.read(Buf, Offset, nToRead);
// Attempt is made to read nToRead bytes, but a smaller number may be read. nRead is the
// actual number of read bytes. The received data is stored from Buf[Offset].
// Synchronous, blocks the thread until input data is available or the IOException is thrown.
// If the connection is broken off (i.e.the socket is closed), throws IOException or returns
// with nRead < 0. Asynchronous reading (non-blocking input) uses other base classes
}
catch (IOException ex) {………………}
if (nRead < 0) {…………………..}

try {
    m_OutStream.write(Buf, Offset, nToWrite);
// Sends nToWrite bytes from Buf[Offset]. Synchronous, blocks until the end of sending
// or the IOException is thrown. Asynchronous writing (non-blocking output) also exists.
    m_OutStream.flush();
}
catch (IOException ex) {………………}
```

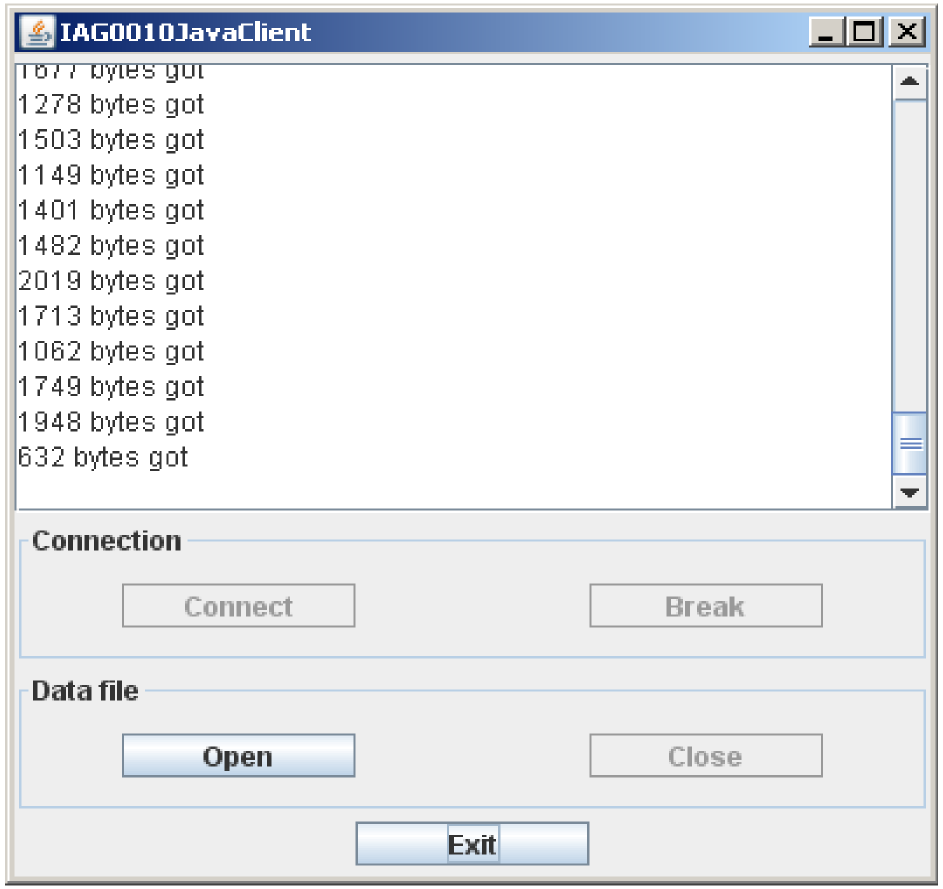
Sockets (3)

```
try {
      m_Socket.close();
// Closes the socket and the input/output streams. Blocked read and write operations will
// throw IOException. To restore the connection you have to create a new socket.
}
catch (IOException ex) {…………}

try {
      m_Socket.shutdownInput();
// Stops reading (read() will return with -1). The socket is not closed.
}
catch (IOException ex) {…………}

try {
      m_Socket.shutdownOutput();
// Disables sending through the socket.
}
catch (IOException ex) {…………}
```

GUI fundamentals (1)



```
1677 bytes got
1278 bytes got
1503 bytes got
1149 bytes got
1401 bytes got
1482 bytes got
2019 bytes got
1713 bytes got
1062 bytes got
1749 bytes got
1948 bytes got
632 bytes got
```

The window that contains the GUI is the frame.
The frame contains a title bar and a panel.
The title bar contains the application icon, the title and three buttons (for minimizing, maximizing and closing).
The panel contains controls. Here we have the following of them:
1. Five buttons.
2. Two inner panels with titles.
3. One text area with vertical scroll bar.

To implement a GUI we have to create a frame as an object of class JFrame or as an object of class derived from JFrame. Similarly, we need a panel (mostly as an object of class derived from JPanel), buttons (mostly as objects of class JButton), etc.

GUI fundamentals (2)

The Java standard classes used for building a GUI are divided into two packages: the java.awt.* and javax.swing.*.

The AWT (abstract Windows toolkit) classes are so-called heavyweight classes because they rely on the underlying operating system (Windows, Linux). It means that although the methods are always the same, their implementation (and consequently the GUI look and feel) depends on the tools provided by operating system.

The Swing classes are lightweight because they are implemented in Java and thus (at last in most cases) guaratee the same look and feel in each environment. They are derived from AWT classes. The class names in Swing start with character 'J', for example there is AWT class Button and Swing class JButton.

Usually a GUI relies on the Swing classes, but directly or indirectly applies the AWT classes too.

GUI fundamentals (3)

The simplified hierarchy of GUI classes is as below:

```
                   Component
                       |
Window──────────Container──────────JComponent
  |                                     |
 Frame         ─────────────────────────────────────
  |           |          |           |          |
JFrame  JPanel  JButton  JTextArea  JLabel …..
```

The component is an object with graphical representation. You can draw the component on the screen and interact with it. The container is a component that can hold inside it another components (including another containers).

Event handling (1)

When the user interacts (clicks, presses a key, changes the dimensions, etc.) with a GUI, an event occurs. As the reaction, the JVM generates an object of class AWTEvent or its subclasses like ActionEvent, ComponentEvent, KeyEvent, FocusEvent, etc. (package java.awt.event.*).
The component on which the event occurred is the event source. For example, if we have

```
JButton m_Button = new JButton("Click me");
```

and the user clicks this button, the JVM generates an object of class ActionEvent with source m_Button.
The event object is sent to an object called listener. The listener waits until it receives an event and then processes it.
For action events, the listener must be an object of class implementing interface ActionListener. For example:

```
public class m_ButtonActionListener implements ActionListener {
        public void actionPerformed(ActionEvent e) {
                                // interface ActionListener has only one method: actionPerformed
                System.out.println("Click detected");
        }
}
```

Event handling (2)

To bind the listener to the event source, write

```
m_Button.addActionListener(new m_ButtonActionListener());
```

To simplify the code, use inner classes, for example:

```
public class GUIMainPanel extends JPanel {
    JButton m_Button = new JButton("Click me");
    …………………………….
    public GUIMainPanel() {
        m_Button.addActionListener(new m_ButtonActionListener());
        ………………………….
    }
    private class m_ButtonActionListener implements ActionListener {
                        // inner classes are usually private
        public void actionPerformed(ActionEvent e) {
            System.out.println("Click detected");
    }
    ………………………………….
}
```

Event handling (3)

The other way to simplify the code is to use anonymous inner classes:

```
m_Button.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e) {
                System.out.print("Click detected");
        }
});
```

Here we define a new class that has no name and an object from that class. We even do not write that this class implements interface ActionListener - the compiler is able to guess it.

Working with mouse:

1. Mouse events: mouse clicked, mouse cursor entered a component, mouse cursor exited a component, mouse button pressed, mouse button released.
2. Mouse motion events: mouse moved on a component, mouse dragged on a component.
3. Mouse wheel events: wheel rotated.

Interfaces for listeners: MouseListener, MouseMotionListener, MouseWheelListener.
If we want a GUI component to react to some of the mouse events, we need to implement the corresponding interface and bind it to the component.
The MouseAdapter class implements all the three mouse interfaces with empty functions. Therefore, instead of implementing interfaces we may extend the adapter.

Event handling (4)

```
m_Button.addMouseListener(new MouseAdapter() {
    public void mouseEntered(MouseEvent e) {
        m_Button.setBackground(Color.RED);
// Class Color belongs to package java.awt.*.
// RED, LIGHT_GRAY, BLUE, GREEN etc. are its static attributes
    }
    public void mouseExited(MouseEvent e) {
        m_Button.setBackground(Color.LIGHT_GRAY);
    }
});
```

Window events: opened, closed, closing, iconified, etc. Necessary for exiting:

```
JFrame frame = new JFrame();
frame.addWindowListener(new WindowAdapter() {
    public void windowClosing(WindowEvent e) {
        // when the user exits the application by closing the GUI frame
        ………………………….. // operations needed before exit
        System.exit(0); // terminates the JVM
    }
});
```

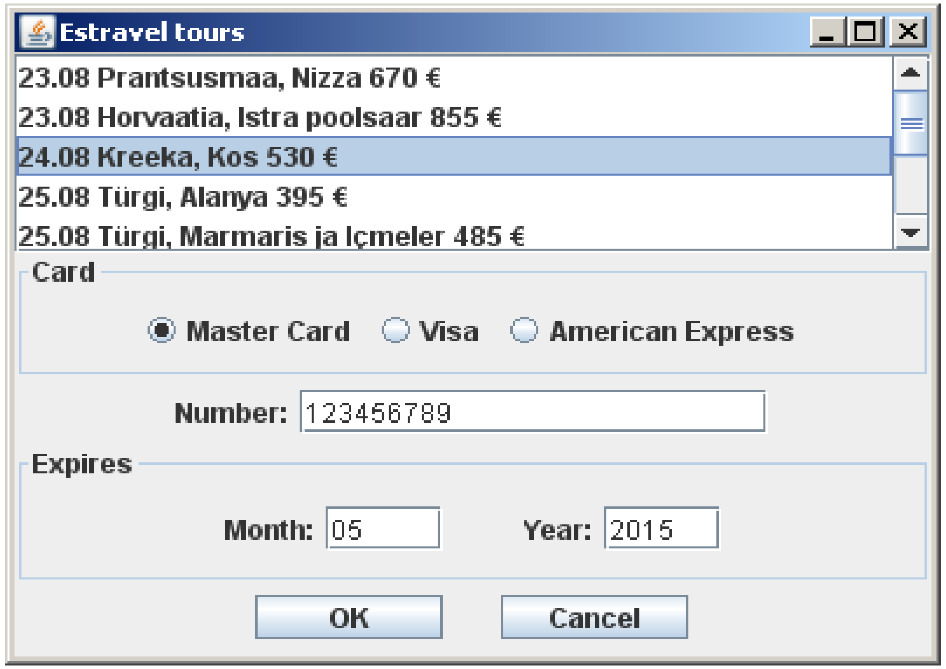
Frame, panel and controls (1)

The GUI main window is an object of class JFrame or its subclass.
JFrame m_Frame = new JFrame("*title text*");
m_Frame.setSize(*width*, *height*); // by default 0*0, unit is pixel
m_Frame.setLocation(*upper_left_x*, *upper_left_y*); // by default (0, 0), unit is pixel
m_Frame.setVisible(true); // necessary call
Do not forget that the screen dimensions depend on the computer hardware.

The GUI runs in its own JVM-created thread. Even if the main() and all the other threads have stopped, the GUI thread may continue running and the application does not exit.
If you want the entire application to terminate when the frame is closed, implement the reaction to windowClosing event. Alternative:
m_Frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE); // static attribute
Useful when there are no operations preceding the exit (ask the user to confirm, close the connections, etc.). The other options:
JFrame.HIDE_ON_CLOSE // hide the GUI window but not terminate the thread
JFrame.DISPOSE_ON_CLOSE // terminate the GUI thread but not the application
JFrame.DO_NOTHING_ON_CLOSE // neither terminate nor hide
In most cases the panel inside the frame is an object of class derived from JPanel.
public class MainPanel extends JPanel {……………………….}
m_Frame.add(new MainPanel());

Frame, panel and controls (2)

The controls are usually the attributes of panel.



23.08 Prantsusmaa, Nizza 670 €
23.08 Horvaatia, Istra poolsaar 855 €
24.08 Kreeka, Kos 530 €
25.08 Türgi, Alanya 395 €
25.08 Türgi, Marmaris ja Içmeler 485 €

Card

[✓ Master Card] Visa American Express

Number: 123456789

Expires

Month: 05 Year: 2015

```
JButton m_OKButton = new JButton("OK");
JButton m_CancelButton =
            new JButton("Cancel");
JLabel m_NumberLabel =
            new JLabel("Number:");
JLabel m_MonthLabel =
            new JLabel("Month:");
JLabel m_YearLabel =
            new JLabel("Year:");
JScrollPane m_ListScrollPane;

JList m_List;
JTextField m_CardNumber = new JTextField();
JTextField m_MonthValue = new JTextField();
JTextField m_YearValue = new JTextField();
JPanel m_CardPanel = new JPanel();
JPanel m_ExpiresPanel = new JPanel();
JRadioButton m_MasterCard = new JRadioButton("Master Card");
JRadioButton m_Visa = new JRadioButton("Visa");
JRadioButton m_AmericanExpress = new JRadioButton("American Express");
```

Frame, panel and controls (3)

1. Buttons:

```
JButton m_Button = new JButton("Title text");
Dimension d = new Dimension(width, height); // in pixels, import java.awt.*
```

Size hints that the layout managers may or may not respect:

```
m_Button.setPreferredSize(d);
m_Button.setMinimumSize(d);
m_Button.setMaximumSize(d);
m_Button.addActionListener(new ActionListener() {
      public void actionPerformed(ActionEvent e) {………………}
});
```

2. Radio buttons must be put into a group:

```
JRadioButton m_RadioButton1 = new JRadioButton("Title text");
……………………………………………………………..
ButtonGroup RadioGroup = new ButtonGroup();
RadioGroup.add(m_RadioButton1 );
……………………………………..
m_RadioButton1.setSelected(true); // initialization
if (m_RadioButton1.isSelected()); // for example, in OK button action processing method
      ………………….
else if …………….
```

Frame, panel and controls (4)

3. Check box:

```
JCheckBox m_CheckBox = new JCheckBox("Title text");
m_CheckBox.setSelected(true); // initialization, by default false
if (m_CheckBox.isSelected()) // for example, in OK button action processing method
    ……………………………..
```

4. Text field (one line of text):

```
JTextField m_TextField = new JTextField("Initial text");
m_ TextField.setPreferredSize(new Dimension(width, height)); // as for JButton
m_TextField.setEditable(false); // by default true
m_TextField.setText("Text to show");
String UsersText = m_TextField.getText();
                                        // for example, in OK button action processing method
```

Action event is generated when the user has pressed ENTER.

5. Text area (several lines of text):

```
JTextArea m_TextArea = new JTextArea("Initial text");
JScrollPane m_ScrollPane = new JScrollPane(m_TextArea);
```

To get a scroll pane with unchangeable dimensions, set all the 3 sizes (min, max and preferred). Otherwise the layout manager sets the dimensions itself. Furthermore, the scroll pane will expand and shrink to fill its parent panel.

```
m_TextArea.append("Text to append to the end");
```

Frame, panel and controls (5)

6. List:

```
String ListElements[] = {………………..};
JList m_List = new JList(ListElements);
JScrollPane m_ScrollPane = new JScrollPane(m_List);
m_List.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
// or SINGLE_INTERVAL_SELECTION or MULTIPLE_INTERVAL_SELECTION
m_List.setSelectedIndex(Initial_selection);
m_List.addListSelectionListener(new ListSelectionListener() {
    public void valueChanged(ListSelectionEvent e) { // import javax.swing.event.*;
        if (!e.getValueIsAdjusting()) // true if deselecting, false if selecting
    // Selecting a row is always preceded by deselecting of the previously selected row.
    // Therefore the list selection event is generated twice.
            System.out.println(m_List.getSelectedIndex());
    }
});
```

7. Label (typically used as identifiers for other components):

```
JLabel m_Label = new JLabel("label text");
```

Layout (1)

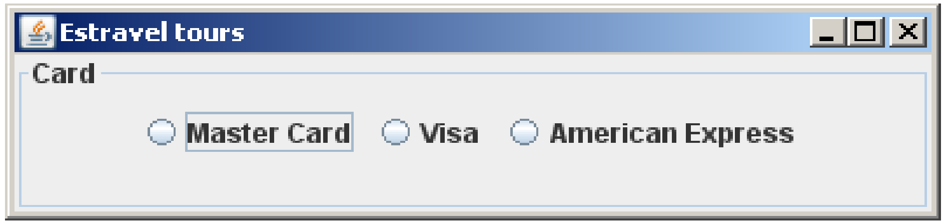
1. Flow layout:

```
JPanel m_CardPanel = new JPanel();
m_CardPanel.setLayout(new FlowLayout(alignment, horizontal_gap, vertical_gap));
```

The components are laid out line-by-line starting from the panel left upper corner (like words in a text editor). When a line is filled, the next component gets its place at the beginning of the next row.

Alignment may be FlowLayout.LEFT, FlowLayout.RIGHT or FlowLayout.CENTER (exactly as words in a text editor)

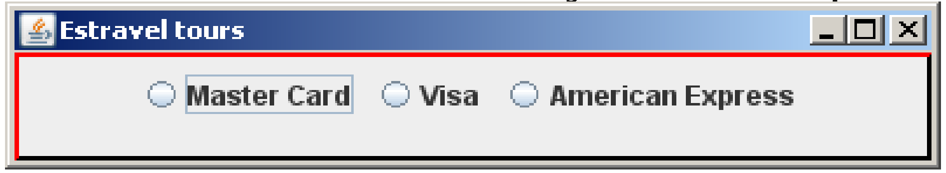
The gaps are between components as well as between components and the panel border.



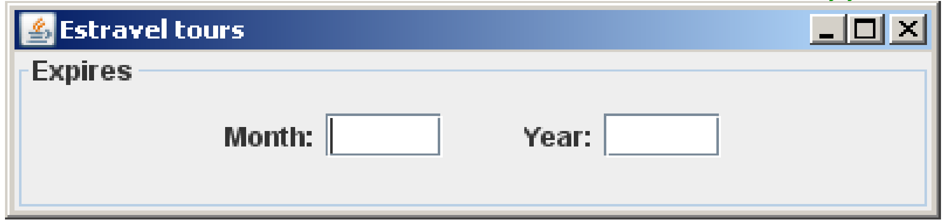
```
m_CardPanel.setBorder(BorderFactory.createTitledBorder("Card"));
m_CardPanel.setLayout(new FlowLayout(FlowLayout.CENTER, 5, 5));
m_CardPanel.add(m_MasterCard);
m_CardPanel.add(m_Visa);
m_CardPanel.add(m_AmericanExpress);
```

Layout (2)

By default, a panel has no border. To set a border (12 types are defined), use the static methods of BorderFactory. For example:



```
m_CardPanel.setBorder(BorderFactory.createBevelBorder(BevelBorder.LOWERED,
            Color.BLACK, Color.RED));
```



```
JPanel ExpireLeft = new JPanel();
JPanel ExpireRight = new JPanel();
JPanel m_ExpiresPanel = new JPanel();
```

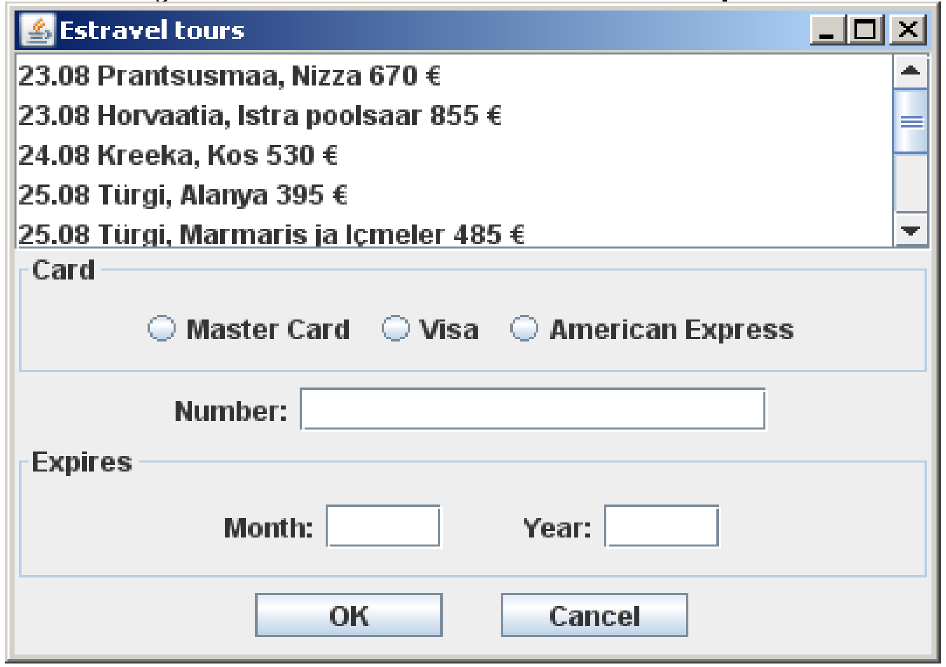
```
ExpireLeft.setLayout(new FlowLayout(FlowLayout.RIGHT, 5, 5)); // auxiliary panel
ExpireLeft.add(m_MonthLabel);
ExpireLeft.add(m_MonthValue);
ExpireRight.setLayout(new FlowLayout(FlowLayout.LEFT, 5, 5)); // auxiliary panel
ExpireRight.add(m_YearLabel);
ExpireRight.add(m_YearValue);
m_ExpiresPanel.setBorder(BorderFactory.createTitledBorder("Expires"));
m_ExpiresPanel.setLayout(new FlowLayout(FlowLayout.CENTER, 25, 5));
m_ExpiresPanel.add(ExpireLeft); // panel is also a component
m_ExpiresPanel.add(ExpireRight);
```

Layout (3)

2. Box layout

```
JPanel Panel = new JPanel();
Panel.setLayout(new BoxLayout(Panel, BoxLayout.Y_AXIS)); // or X_AXIS
```

BoxLayout either stacks its components on top of each other or places them in a row.



Estravel tours

23.08 Prantsusmaa, Nizza 670 €
23.08 Horvaatia, Istra poolsaar 855 €
24.08 Kreeka, Kos 530 €
25.08 Türgi, Alanya 395 €
25.08 Türgi, Marmaris ja Içmeler 485 €

```
Panel. add(m_ListScrollPane);
Panel. add(m_CardPanel);
Panel. add(NumberPanel);
Panel. add(m_ExpiresPanel);
Panel. add(ButtonPanel);
```

Swing and threads (1)

Any GUI program has several JVM-created threads. To avoid synchronization problems, all the calls to Swing components must occur only from one of them: the event dispatch thread.

Calls to Swing components initiated by user's actions are always from the event dispatch thread. Calls from the application non-Swing threads must be written as follows:

```
SwingUtilities.invokeLater(new Runnable() {
        public void run() {
                ……………….. // operations accessing GUI components
        }
});
```

Here we define a new anonymous class implementing interface Runnable and an object from that class. The object is passed to the event dispatch thread which executes the run(). The invokeLater method returns immediately and the thread that called it can continue. The run() is executed at some time in the future - you cannot know when it will actually work.

Swing and threads (2)

The alternative is:

```
try {
      SwingUtilities.invokeAndWait(new Runnable() {
              public void run() {
                      …………….. // operations accessing GUI components
              }
      });
}
catch (InterruptedException ex1) {…………………} // waiting interrupted
catch (InvocationTargetException ex2) {……………..} // errors or exceptions in run()
```

The thread that called invokeAndWait() blocks until the event dispatch thread has executed the run() method.

Do not call invokeAndWait() from a method that itself runs in the event dispatch thread. If you are not sure, check:

```
if (!SwingUtilities.isEventDispatchThread()) {…….} // call to invokeAndWait()
```

Advise: use the invokeAndWait() to read values from Swing components. Use it also when you need to be sure that when the thread continues, the information you want to write on the screen is written and visible. Otherwise use invokeLater().

Message boxes

JOptionPane.showMessageDialog(*null*, "*Message text*", "*Box title text*", *Type*);
The type determines the box icon : JOptionPane.ERROR_MESSAGE,
JOptionPane. INFORMATION_MESSAGE, JOptionPane. PLAIN_MESSAGE,
JOptionPane. QUESTION_MESSAGE, JOptionPane. WARNING_MESSAGE
The box has only the OK button.

JOptionPane.showConfirmDialog(*null*, "*Message text*", "*Box title text*", *Type*);
The type determines the buttons number and titles : JOptionPane.YES_NO_OPTION,
JOptionPane. YES_NO_CANCEL_OPTION, JOptionPane. OK_CANCEL_OPTION
The output is JOptionPane.YES_OPTION, JOptionPane.NO_OPTION,
JOptionPane.OK_OPTION or JOptionPane.CANCEL_OPTION.

File JavaGUI.java

```
package javagui;
import javax.swing.*;
import java.awt.event.*;

public class JavaGUI {
    static private MainPanel m_MainPanel;
    public static void main(String[] args) {
        try {
            SwingUtilities.invokeAndWait(new Runnable() {
                public void run() {
                    JFrame frame = new JFrame("Estravel tours");
                    frame.setSize(400, 300);
                    frame.setLocation(100, 100);
                    m_MainPanel = new MainPanel();
                    frame.add(m_MainPanel);
                    frame.addWindowListener(new WindowAdapter() {
                        public void windowClosing(WindowEvent e) {
                            System.exit(0);
                        }
                    });
                    frame.setVisible(true);
                    }
                });
            }
        catch (Exception e) {
                System.out.println("Failed to create main panel");
                return;
        }
    }
}
```

File MainPanel.java

```
package javagui;
import java.awt.*;
import javax.swing.*;
import javax.swing.event.*;

public class MainPanel extends javax.swing.JPanel {
    private JButton m_OKButton = new JButton("OK");
    private JButton m_CancelButton = new JButton("Cancel");
    private JLabel m_NumberLabel = new JLabel("Number:");
    private JLabel m_MonthLabel = new JLabel("Month:");
    private JLabel m_YearLabel = new JLabel("Year:");
    private JScrollPane m_ListScrollPane;
private JList m_List;
private JTextField m_CardNumber = new JTextField();
private JTextField m_MonthValue = new JTextField();
private JTextField m_YearValue = new JTextField();
private JPanel m_CardPanel = new JPanel();
private JPanel m_ExpiresPanel = new JPanel();
private JRadioButton m_MasterCard = new JRadioButton("Master Card");
private JRadioButton m_Visa = new JRadioButton("Visa");
private JRadioButton m_AmericanExpress = new JRadioButton("American Express");

public MainPanel() {
    String Tours[] = {
        "23.08 Prantsusmaa, Nizza 670 EUR",
        "23.08 Horvaatia, Istra poolsaar 855 EUR",
        "24.08 Kreeka, Kos 530 EUR",
        "25.08 Hispaania, Mallorca, Santa Ponsa 495 EUR",
        "25.08 Kreeka, Rhodos 585 EUR",
        "25.08 Kreeka, Lefkas 715 EUR"
    };
    m_List = new JList(Tours);
    m_List.addListSelectionListener(new ListSelectionListener() {
        public void valueChanged(ListSelectionEvent e) {
            if (!e.getValueIsAdjusting())
                System.out.println(m_List.getSelectedIndex());
        }
    });
    m_List.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
    m_ListScrollPane = new JScrollPane(m_List);

    m_CardPanel.setBorder(BorderFactory.createTitledBorder("Card"));
    m_CardPanel.setLayout(new FlowLayout(FlowLayout.CENTER, 5, 5));
    m_CardPanel.add(m_MasterCard);
    m_CardPanel.add(m_Visa);
    m_CardPanel.add(m_AmericanExpress);
    ButtonGroup cards = new ButtonGroup();
    cards.add(m_MasterCard);
    cards.add(m_Visa);
    cards.add(m_AmericanExpress);

    JPanel NumberPanel = new JPanel();
    NumberPanel.setLayout(new FlowLayout(FlowLayout.CENTER));
    NumberPanel.add(m_NumberLabel);
    m_CardNumber.setPreferredSize(new Dimension(200, 20));
    NumberPanel.add(m_CardNumber);
```

```
JPanel ExpireLeft = new JPanel();
JPanel ExpireRight = new JPanel();
ExpireLeft.setLayout(new FlowLayout(FlowLayout.RIGHT, 5, 5));
ExpireLeft.add(m_MonthLabel);
m_MonthValue.setPreferredSize(new Dimension(50, 20));
ExpireLeft.add(m_MonthValue);
ExpireRight.setLayout(new FlowLayout(FlowLayout.LEFT, 5, 5));
ExpireRight.add(m_YearLabel);
m_YearValue.setPreferredSize(new Dimension(50, 20));
ExpireRight.add(m_YearValue);
m_ExpiresPanel.setBorder(BorderFactory.createTitledBorder("Expires"));
m_ExpiresPanel.setLayout(new FlowLayout(FlowLayout.CENTER, 25, 5));
m_ExpiresPanel.add(ExpireLeft);
m_ExpiresPanel.add(ExpireRight);

JPanel ButtonPanel = new JPanel();
ButtonPanel.setLayout(new FlowLayout(FlowLayout.CENTER, 25, 5));
m_OKButton.setPreferredSize(new Dimension(80, 20));
ButtonPanel.add(m_OKButton);
m_CancelButton.setPreferredSize(new Dimension(80, 20));
ButtonPanel.add(m_CancelButton);
```

```
    setLayout(new BoxLayout(this, BoxLayout.Y_AXIS));
    add(m_ListScrollPane);
    add(m_CardPanel);
    add(NumberPanel);
    add(m_ExpiresPanel);
    add(ButtonPanel);
}
```

Applets (1)

Applet is a Java application that is accessed on an Internet server, transported over the Internet, automatically installed on the client computer and runs as a part of web page.
The HTML page containing an applet must have the following tag:
<applet code = "*AppletMain*.class" width = *applet_window_width* height = *applet_window_height*> </applet>
The width and height are in pixels. The applet main class must be inherited from class JApplet (Swing) or Applet (AWT).
Starting from HTML version 4.01 the applet tag is deprecated and should be replaced by tag
<object data = "*AppletMain*.class" width = *applet_window_width* height = *applet_window_height*> </object>
However, not all the browsers support the object tag. Therefore the applet tag is more reliable. It is also recommended by Oracle.
Example:
<applet code = "AppletExample.class" width = 400 height = 300> </applet>
If the applet consists several classes, pack them into a jar file:
<applet archive = "*AppletJar*.jar" code = "*AppletMain*.class"
width = *applet_window_width* height = *applet_window_height*> </applet>

Applets (2)

The applet has no main() method. The browser controls the applet by 4 functions inherited from JApplet:

```
public class AppletExample extends JApplet {
public void init() {……………..} // the first method the browser calls
public start() {……………..} // the browser calls it when the applet starts to work
                                    // or resumes its work
public stop() {……………..} // the browser calls it when the work of applet is suspended
public destroy() {……………..} // called when the browser terminates the applet
}
```

In JApplet those methods are empty. Almost in any applet the programmer should override the init() method. Overriding the other methods is seldom needed.

```
public void init() {
    try {
        SwingUtilities.invokeAndWait(new Runnable() { // do not use invokeLater()
            public void run() {
                …………………. // define GUI components and set layout
            }
        });
    }
    catch (Exception ex) {………………….}
}
```

Applets (3)

Applets do not have frames. The main panel is also not needed:
`Component – Container – Panel – Applet – JApplet`

Applet may have parameters:
<applet code = "*AppletMain*.class" width = *applet_window_width* height = *applet_window_height*>
<param name = *ParameterName_1* value = *ParameterValue_1*>
……………………………………………………………………….
<param name = *ParameterName_n* value = *ParameterValue_n*>
 </applet>
Parameters are read and converted in start():

```
public void start() {
        String Par;
        Par = getParameter(" ParameterName_1");
        if (Par != null) { // Par is now the string equivalent with ParameterValue_1
        ……………………..
        }
        ……………………………………….
}
```

Applets (4)

Due to the security restirictions an applet cannot:

1. Read data stored in the client computer files.
2. Modify, add or delete data stored in the client computer files.
3. Create new files on the client computer.
4. Run programs on the client computer.
5. Read the client computer system settings (except the Java and operating system versions)
6. Change the system settings on the client computer.