Java RMI Tutorial

Vania Marangozova-Martin

Vania.Marangozova-Martin@imag.fr

ids.forge.imag.fr

Let us start with Java…

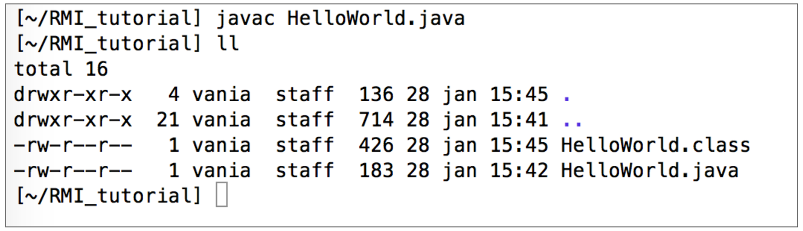
- There is the Java language
  - OO-oriented
- There are the Java tools
  - including the compiler: javac
- There is the JVM = Java Virtual Machine
- There is the JDK = Java Development Kit
- There is the JRE = Java Runtime Environment

Your First Java Program

```
public class HelloWorld {

    public static void main(String[] args) {
        // Prints "Hello, World" to the terminal window.
        System.out.println("Hello, World");
    }

}
```

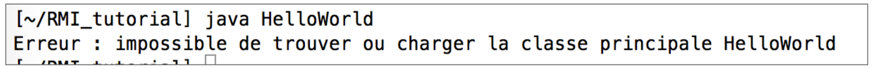
- This should be put in the file `HelloWorld.java`
  - i.e the name of the class and the name of the file should be the same

Compiling the program



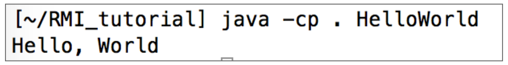
```
[~/RMI_tutorial] javac HelloWorld.java
[~/RMI_tutorial] ll
total 16
drwxr-xr-x   4 vania  staff  136 28 jan 15:45 .
drwxr-xr-x  21 vania  staff  714 28 jan 15:41 ..
-rw-r--r--   1 vania  staff  426 28 jan 15:45 HelloWorld.class
-rw-r--r--   1 vania  staff  183 28 jan 15:42 HelloWorld.java
[~/RMI_tutorial] 
```

Executing the program



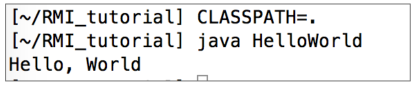
```
[~/RMI_tutorial] java HelloWorld
Erreur : impossible de trouver ou charger la classe principale HelloWorld
```

- classpath: say where the classes are
  - use java –cp <directories;jarfiles>



```
[~/RMI_tutorial] java -cp . HelloWorld
Hello, World
```

  - or set the environment variable CLASSPATH



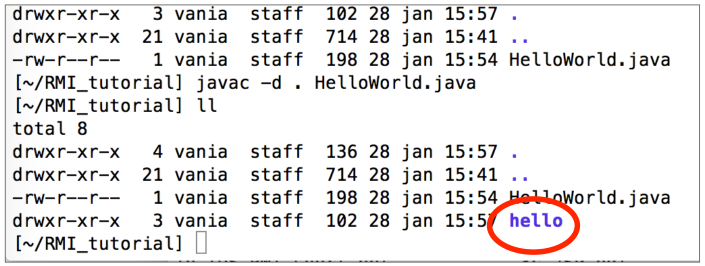
```
[~/RMI_tutorial] CLASSPATH=.
[~/RMI_tutorial] java HelloWorld
Hello, World
```

Java Packages

```
package hello;

public class HelloWorld {

    public static void main(String[] args) {
        // Prints "Hello, World" to the terminal window.
        System.out.println("Hello, World");
    }
}
```

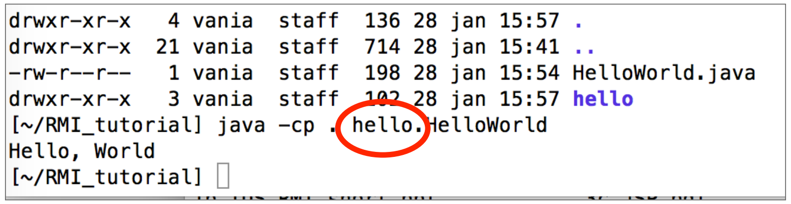
- *"A package can be defined as a grouping of related types (classes, interfaces, enumerations and annotations) providing access protection and name space management."*

Compiling the program



```
drwxr-xr-x   3 vania  staff  102 28 jan 15:57 .
drwxr-xr-x  21 vania  staff  714 28 jan 15:41 ..
-rw-r--r--   1 vania  staff  198 28 jan 15:54 HelloWorld.java
[~/RMI_tutorial] javac -d . HelloWorld.java
[~/RMI_tutorial] ll
total 8
drwxr-xr-x   4 vania  staff  136 28 jan 15:57 .
drwxr-xr-x  21 vania  staff  714 28 jan 15:41 ..
-rw-r--r--   1 vania  staff  198 28 jan 15:54 HelloWorld.java
drwxr-xr-x   3 vania  staff  102 28 jan 15:57 hello
[~/RMI_tutorial] 
```

Executing the program



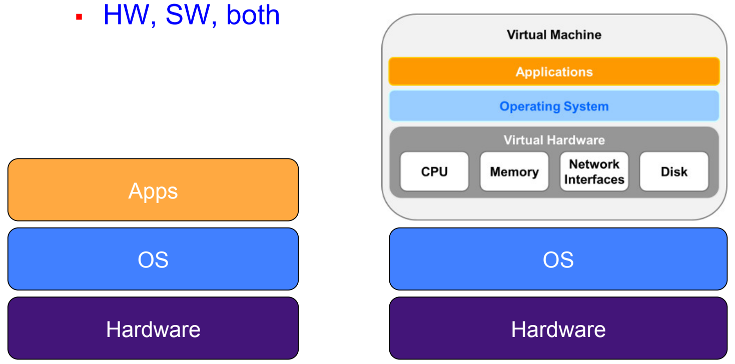
```
drwxr-xr-x   4 vania  staff  136 28 jan 15:57 .
drwxr-xr-x  21 vania  staff  714 28 jan 15:41 ..
-rw-r--r--   1 vania  staff  198 28 jan 15:54 HelloWorld.java
drwxr-xr-x   3 vania  staff  102 28 jan 15:57 hello
[~/RMI_tutorial] java -cp . hello.HelloWorld
Hello, World
[~/RMI_tutorial] 
```

On IDEs

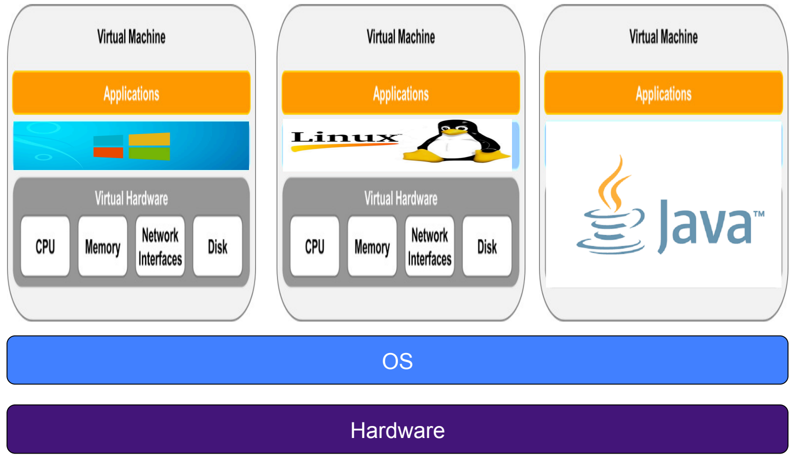
- IDE = Integrated Development Environment
  - Eclipse
  - NetBeans
  - …
- You are free to use them
- They put additional complexity
  - you need to understand Java
  - and the IDE

The JVM

- What is a virtual machine?
  - Imitates a particular computer system
    - HW, SW, both



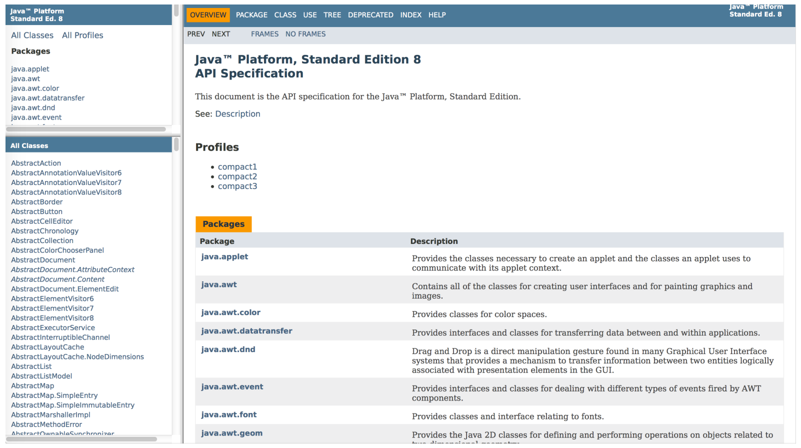
The JVM (2)



The JVM (3)

- The machine code for Java applications is called bytecode
- This is what is obtained when compiling Java

The JDK (development)



Java™ Platform
Standard Ed. 8

All Classes All Profiles

Packages

java.applet
java.awt
java.awt.color
java.awt.datatransfer
java.awt.dnd
java.awt.event

All Classes

AbstractAction
AbstractAnnotationValueVisitor6
AbstractAnnotationValueVisitor7
AbstractAnnotationValueVisitor8
AbstractBorder
AbstractButton
AbstractCellEditor
AbstractChronology
AbstractCollection
AbstractColorChooserPanel
AbstractDocument
AbstractDocument.AttributeContext
AbstractDocument.Content
AbstractDocument.ElementEdit
AbstractElementVisitor6
AbstractElementVisitor7
AbstractElementVisitor8
AbstractExecutorService
AbstractInterruptibleChannel
AbstractLayoutCache
AbstractLayoutCache.NodeDimensions
AbstractList
AbstractListModel
AbstractMap
AbstractMap.SimpleEntry
AbstractMap.SimpleImmutableEntry
AbstractMarshallerImpl
AbstractMethodError
AbstractOwnableSynchronizer

OVERVIEW PACKAGE CLASS USE TREE DEPRECATED INDEX HELP — Java™ Platform Standard Ed. 8

PREV NEXT FRAMES NO FRAMES

**Java™ Platform, Standard Edition 8
API Specification**

This document is the API specification for the Java™ Platform, Standard Edition.

See: Description

Profiles

- compact1
- compact2
- compact3

Packages

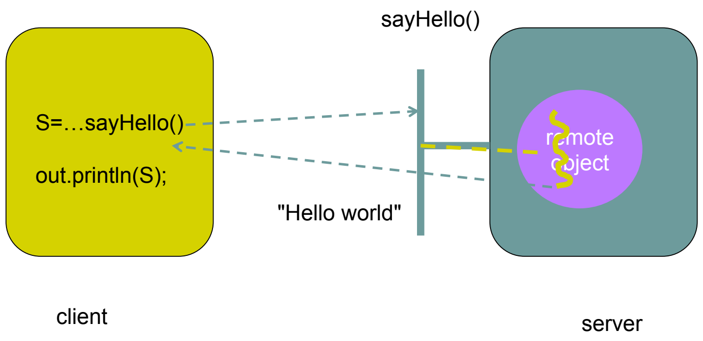
Package	Description
java.applet	Provides the classes necessary to create an applet and the classes an applet uses to communicate with its applet context.
java.awt	Contains all of the classes for creating user interfaces and for painting graphics and images.
java.awt.color	Provides classes for color spaces.
java.awt.datatransfer	Provides interfaces and classes for transferring data between and within applications.
java.awt.dnd	Drag and Drop is a direct manipulation gesture found in many Graphical User Interface systems that provides a mechanism to transfer information between two entities logically associated with presentation elements in the GUI.
java.awt.event	Provides interfaces and classes for dealing with different types of events fired by AWT components.
java.awt.font	Provides classes and interface relating to fonts.
java.awt.geom	Provides the Java 2D classes for defining and performing operations on objects related to

The JRE (execution)

- Compiled classes + tools to run Java programs

Java RMI

- Let us make "Hello World" distributed…



The Interface

Remote object – Interface definition

```
import java.rmi.*;


public interface Hello
    extends Remote {

    // A method provided by the
    // remote object
    public String sayHello()
          throws RemoteException;

}
```

Hello.java

The remote object implementation

Remote object – Interface definition

```
import java.rmi.*;


public interface Hello
    extends Remote {

    // A method provided by the
    // remore object
    public String sayHello()
           throws RemoteException;

}
```

Hello.java

Remote object – Class implementation

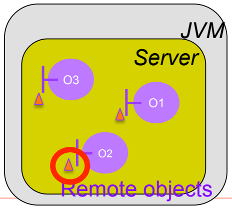
```
import java.rmi.*;
import java.rmi.server.*;

public  class HelloImpl
    implements Hello {

    private String message;

    public HelloImpl(String s) {
           message = s ;
    }

    public String sayHello ()
           throws RemoteException {
           return message ;
    }

}
```

HelloImpl.java

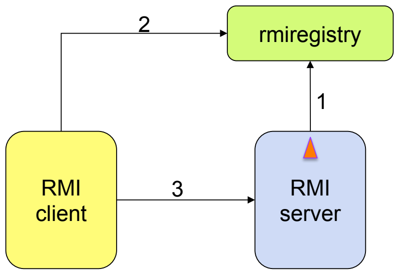
The server

HelloServer.java

```
import java.rmi.*;
import java.rmi.server.*;
import java.rmi.registry.*;

public class HelloServer {

  public static void  main(String [] args){

     try {
        // Create a Hello remote object
       HelloImpl h = new HelloImpl ("Hello world !");
       Hello h_stub = (Hello) UnicastRemoteObject.exportObject(h, 0);
…
```



The server



```
import java.rmi.*;
import java.rmi.server.*;
import java.rmi.registry.*;

public class HelloServer {

  public static void  main(String [] a

     try {
        // Create a Hello remote objec
       HelloImpl h = new HelloImpl ("Hello world !");
       Hello h_stub = (Hello) UnicastRemoteObject.exportObject(h, 0);

       // Register the remote object in RMI registry with a given identifier
       Registry registry= LocateRegistry.getRegistry();
       registry.bind("Hello1", h_stub);

       System.out.println ("Server ready");

     } catch (Exception e)  {
        System.err.println("Error on server :" + e) ;
        e.printStackTrace();
     }
```

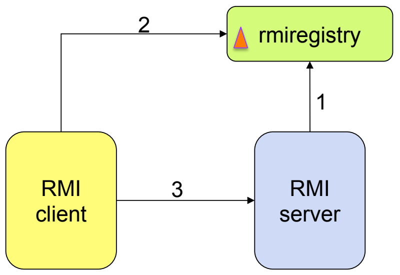
HelloServer.java

The client

```
import java.rmi.*;
import java.rmi.registry.*;

public class HelloClient {
  public static void main(String [] args) {

    try {
      if (args.length < 1) {
       System.out.println("Usage: java HelloClient <rmiregistry host>");
       return;}

    String host = args[0];

    // Get remote object reference
    Registry registry = LocateRegistry.getRegistry(host);
    Hello h = (Hello) registry.lookup("Hello1");

    // Remote method invocation
    String res = h.sayHello();
    System.out.println(res);

    } catch (Exception e)  {
         System.err.println("Error on client: " + e) ;
    }…
```

HelloClient.java

The client



```
import java.rmi.*;
import java.rmi.registry.*;

public class HelloClient {
  public static void main(String [] a

   try {
     if (args.length < 1) {
      System.out.println("Usage: jav
      return;}
```

TRANSPARENCY

```
   String res = h.sayHello();
   System.out.println(res);

   } catch (Exception e)  {
        System.err.println("Error on client: " + e) ;
   }…
```

Development steps for RMI applications

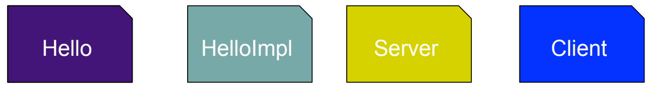
1. Design and implement the components (classes, sources) ✓
2. Compile the java classes
3. Make the classes accessible
4. Run the application

Source compilation

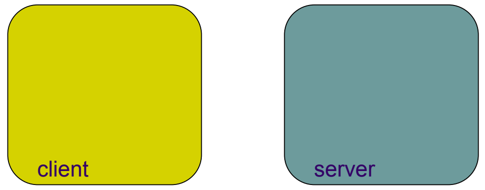
- Directory organization

```
drwxr-xr-x  5 vania  staff  170 29 jan 15:34 .
drwxr-xr-x  5 vania  staff  170 29 jan 14:01 ..
drwxr-xr-x  3 vania  staff  102 29 jan 15:34 classes
drwxr-xr-x  3 vania  staff  102 29 jan 15:34 lib
drwxr-xr-x  6 vania  staff  204 29 jan 15:32 src
```

- Separate compilation



- Why?



Source compilation (2)

- **The interface**

```
javac –d classes –classpath .:classes src/Hello.java
jar cvf lib/Hello.jar classes/Hello.class
```

- **The remote object implementation**

```
javac –d classes –classpath .:classes src/HelloImpl.java
jar cvf lib/HelloImpl.jar classes/HelloImpl.class
```

- **The server side**

```
javac –d classes
      –cp .:classes:lib/Hello.jar:lib/HelloImp.jar
      src/HelloServer.java
```

- **The client side**

```
javac –d classes
      –cp .:classes:lib/Hello.jar
      src/HelloClient.java
```

Development steps for RMI applications

1. Design and implement the components (classes, sources) ✓
2. Compile the java classes ✓
3. Make the classes accessible
4. Run the application

Run the application (locally)

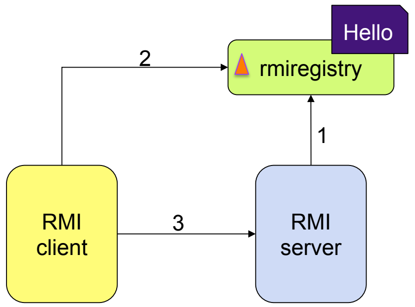
- Start RMI registry
  - `CLASSPATH=where the Hello.jar is`
  - `rmiregistry &`
- Start the server

  -
    ```
    java –classpath .:classes:lib/
    Hello.jar:lib/HelloImpl.jar
    HelloServer
    ```
- Start the client

  -
    ```
    java –classpath .:classes:lib/
    Hello.jar HelloClient localhost
    ```

Run the application (2)

- Start RMI registry
  - `CLASSPATH=where the Hello.jar is`
  - `rmiregistry &`



Development steps for RMI applications

1. Design and implement the components (classes, sources) ✓
2. Compile the java classes ✓
3. Make the classes accessible
4. Run the application ✓

Let us make the example a little bit more complex…

Hello.java

```
import java.rmi.*;

public interface Hello extends Remote {

   // Method provided by the remote object
   public String sayHello() throws RemoteException;
}
```

Second.java

```
import java.rmi.*;

public interface Second extends Remote {

   // Method provided by the remote object
   public String sayHi(int c, Hello o, Person p) throws RemoteException;
}
```

primitive type = Serializable

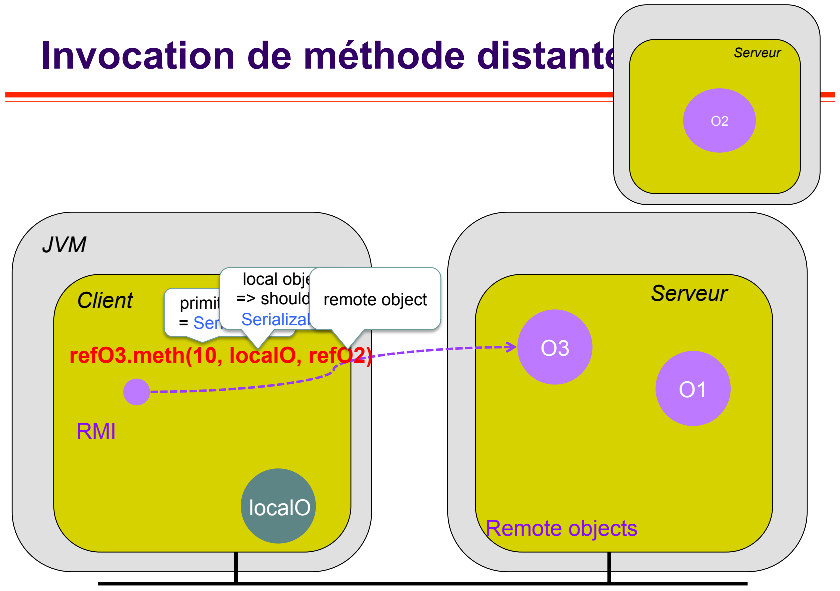
remote object

local object => should be Serializable

Passing objects around in RMI

- Arguments to remote methods or return values from remote methods can be of any type
  - Primitive data types (e.g. int, float, etc.)
  - Remote objects
  - Local objects
- Non remote objects passed to or returned from remote methods must be serializable
  - They must implement the java.io.Serializable interface
- Some object types do not meet any of these criteria; they cannot be passed to or returned from remote methods
  - Most of these objects, such as threads or file descriptors, encapsulate information that makes sense only within a single address space
  - Many of the core classes (e.g. classes in the packages java.lang and java.util) implement the Serializable interface

Invocation de méthode distante



A serializable class

```
public class Person implements Serializable {

private String firstName;
  private String lastName;
  // stupid example for transient
  transient private Thread myThread;

  public Person(String firstName, String lastName) {
    this.firstName = firstName;
    this.lastName = lastName;
    this.myThread = new Thread();
  }
  public String getFirstName() {return firstName;}
  public void setFirstName(String firstName) {this.firstName = firstName;}
  public String getLastName() {return lastName;}
  public void setLastName(String lastName) {this.lastName = lastName;}
  @Override
  public String toString() {
    return "Person [firstName=" + firstName + ", lastName=" + lastName
        + "]";
  }

}
```

The Second remote object implementation

Remote object – Interface definition

```
import java.rmi.*;


public interface Second
    extends Remote {

    // Method provided by the remote
    object
    public
    String sayHi( int c,
                  Hello o,
                  Person p)
    throws RemoteException;
}
```

Second.java

Remote object – Class implementation

```
import java.rmi.*;
import java.rmi.server.*;

public  class SecondImpl
    implements Second {
    public SecondImpl() {super();}

    public String sayHi (int c,
                  Hello o,
                  Person p)
          throws RemoteException {
       return     p.toString()+
                  "says " +
                  o.sayHello() +
                  " " +
                  c +
                  " times!!!";
    }

}
```

SecondImpl.java

The server

```
import java.rmi.*;
import java.rmi.server.*;
import java.rmi.registry.*;

public class SecondServer {

  public static void  main(String [] args){
     try {
        // Create a Hello remote object
       HelloImpl h = new HelloImpl ("Hello world !");
       Hello h_stub =  (Hello) UnicastRemoteObject.exportObject(h, 0);
       SecondImpl s = new SecondImpl ();
       Second s_stub = (Second) UnicastRemoteObject.exportObject(s, 1);

       // Register the remote object in RMI registry with a given identifier
       Registry registry= LocateRegistry.getRegistry();
       registry.bind("Hello1", h_stub);
       registry.bind("second", s_stub);
       System.out.println ("Server ready");

     } catch (Exception e)  {
        System.err.println("Error on server :" + e) ;
        e.printStackTrace();
     }
```

The client

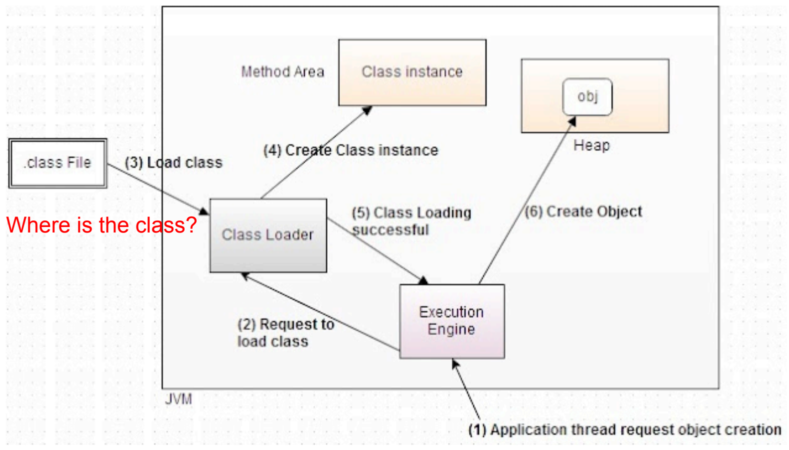
```
import java.rmi.*;
import java.rmi.registry.*;

public class SecondClient {
  public static void main(String [] args) {

    try {
         if (args.length < 1) {
         System.out.println("Usage: java HelloClient <server host>");
         return;}
         String host = args[0];
    // Get remote object reference
         Registry registry = LocateRegistry.getRegistry(host);
         Hello h = (Hello) registry.lookup("Hello1");
         Second s = (Second) registry.lookup("second");
    //Person creation
         Person p = new Person("Vania", "Marangozova");
    // Remote method invocation
         String res = h.sayHello(); System.out.println(res);

         String res2 = s.sayHi(10, h, p); System.out.println(res2);
    } catch (Exception e)  {
         System.err.println("Error on client: " + e) ;
    }…
```

Make the classes accessible…

Source: http://docs.oracle.com/javase/specs/jvms/se7/html/jvms-5.html

- Classes? Accessible?
- What about the classes and the JVM?
  - *The Java Virtual Machine dynamically loads, links and initializes classes and interfaces.*
  - *Loading is the process of finding the binary representation of a class or interface type with a particular name and creating a class or interface from that binary representation.*
  - *Linking is the process of taking a class or interface and combining it into the run-time state of the Java Virtual Machine so that it can be executed.*
  - *Initialization of a class or interface consists of executing the class or interface initialization method <clinit>*

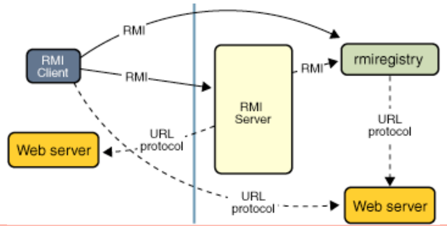
ClassLoader


Method Area
Class instance
obj
Heap
.class File
(3) Load class
(4) Create Class instance
Where is the class?
Class Loader
(5) Class Loading successful
(6) Create Object
Execution Engine
(2) Request to load class
JVM
(1) Application thread request object creation

Credit: http://geekrai.blogspot.fr/2013/05/class-loading-and-unloading-in-jvm.html

Codebase

- Codebase = the place where the JVM may find classes
  - CLASSPATH = local
  - Possibility to have remote classes
    - accessible via the network
    - HTTP or FTP



Références

- **IBM. Introduction to Java programming**

http://www.ibm.com/developerworks/java/tutorials/j-introtojava1/

- **Oracle. Trail: RMI.**

http://docs.oracle.com/javase/tutorial/rmi/

- http://www.securingjava.com/chapter-two/chapter-two-7.html
- https://www.prologin.org/docs/java/technotes/guides/rmi/codebase.html
- http://www.kedwards.com/jini/codebase.html
- https://docs.oracle.com/javase/tutorial/rmi/running.html

END

SecurtyManager

- A *security manager* is an object that defines a security policy for an application.
- This policy specifies actions that are unsafe or sensitive.
- Any actions not allowed by the security policy cause a SecurityException to be thrown.
- An application can also query its security manager to discover which actions are allowed.
- A security manager determines whether downloaded code has access to the local file system or can perform any other privileged operations
- If an RMI program does not install a security manager, RMI will not download classes (other than from the local class path) for objects received as arguments or return values of remote method invocations
- This restriction ensures that the operations performed by downloaded code are subject to a security policy

SecurtyManager (2)

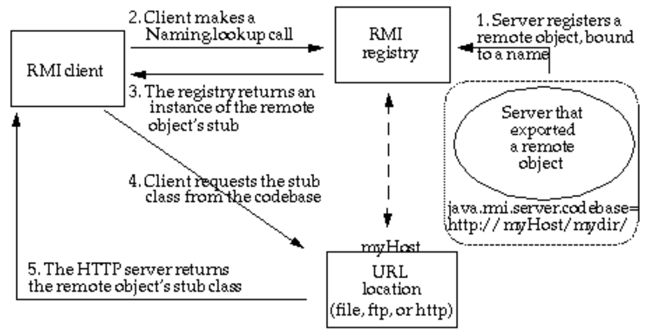
```
SecurityManager sm = System.getSecurityManager();
if (sm != null) context = sm.getSecurityContext();
if (sm != null) sm.checkPermission(permission, context);
```

- Permissions fall into these categories:
  - File, Socket, Net, Security, Runtime, Property, AWT, Reflect, and Serializable.
- The classes managing these various permission categories are
  - `java.io.FilePermission`, `java.net.SocketPermission`, `java.net.NetPermission`, `java.security.SecurityPermission`, `java.lang.RuntimePermission`, `java.util.PropertyPermission`, `java.awt.AWTPermission`, `java.lang.reflect.ReflectPermission`, and `java.io.SerializablePermission`.
- http://docs.oracle.com/javase/7/docs/technotes/guides/security/permissions.html
- http://docs.oracle.com/javase/7/docs/technotes/guides/security/PolicyFiles.html

Exemples de fichiers de securité (3)

- Policy files

```
grant codeBase "file:/home/ann/src/" {
   permission java.security.AllPermission;
};
grant codeBase "file:/home/sysadmin/" {
   permission java.io.FilePermission "/tmp/abc", "read";
};
grant codeBase "file://f:/derby/lib/derby.jar" {
   permission java.lang.RuntimePermission "createClassLoader";
   permission java.util.PropertyPermission "derby.*", "read";
   permission.java.io.FilePermission "${derby.system.home}","read";
   permission java.io.FilePermission "${derby.system.home}${/} -",
   "read,write,delete";
   permission java.util.PropertyPermission
   "derby.storage.jvmInstanceId", "write";
};
```


2. Client makes a Naming.lookup call
RMI registry
1. Server registers a remote object, bound to a name
RMI client
3. The registry returns an instance of the remote object's stub
Server that exported a remote object
java.rmi.server.codebase= http://myHost/mydir/
4. Client requests the stub class from the codebase
myHost
5. The HTTP server returns the remote object's stub class
URL location (file, ftp, or http)