LAB 01

Κατανεμημένα Συστήματα

Java RMI - Remote Method Invocation

01

ΕΙΣΑΓΩΓΗ ΣΤΗ JAVA RMI

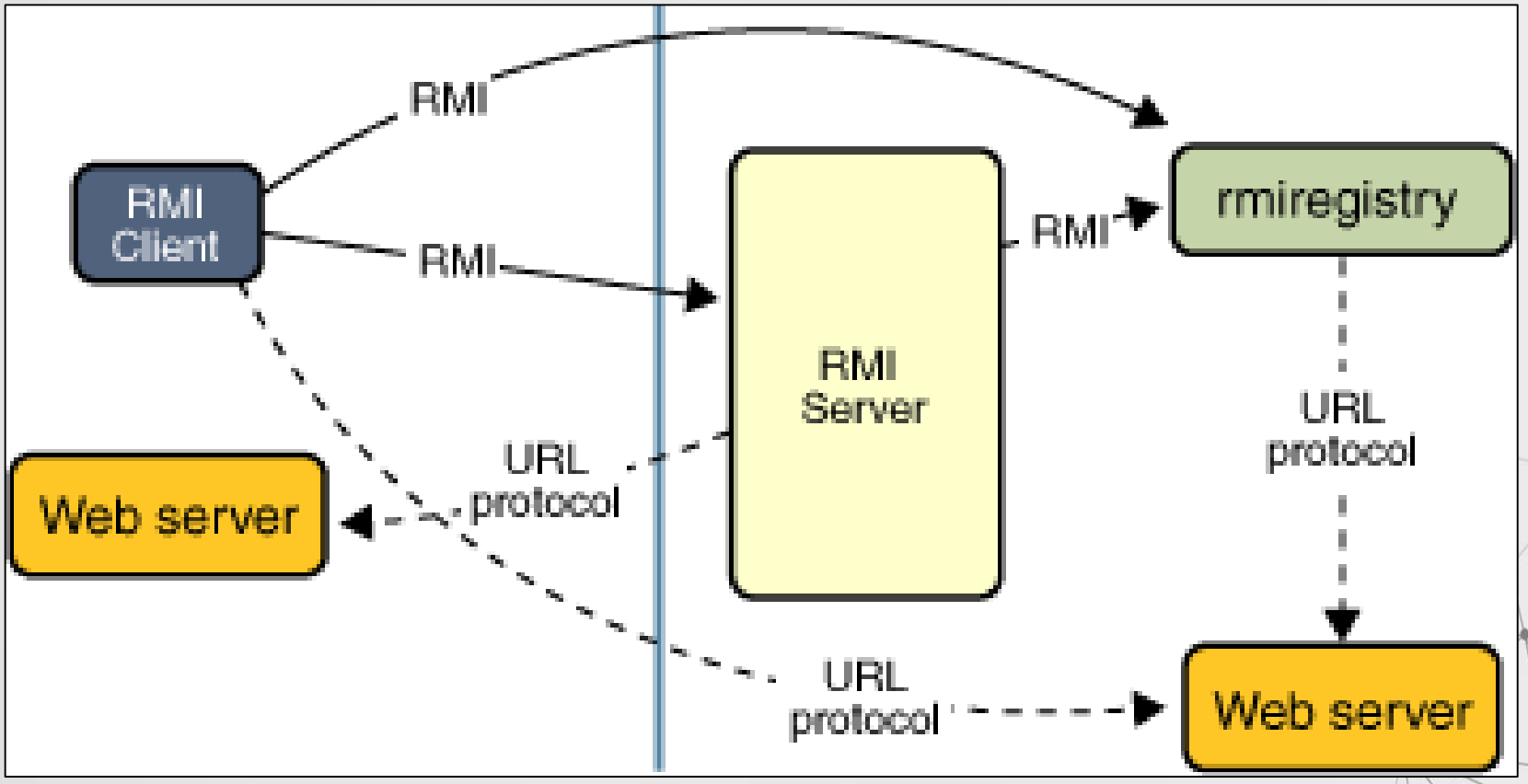
Εφαρμογές Java RMI

- Αποτελούνται από δύο εφαρμογές: server και client.
- Εφαρμογή Server:
  - δημιουργεί τα απομακρυσμένα αντικείμενα.
  - διαμορφώνει την πρόσβαση σε αυτά.
  - δέχεται αιτήσεις πρόσβασης από πελάτες.
- Εφαρμογή Client:
  - αποκτά πρόσβαση σε απομακρυσμένα αντικείμενα του server.
  - καλεί μεθόδους αυτών των αντικείμενων για να υλοποιήσει την εφαρμογή του.
- Το RMI παρέχει το μηχανισμό για την επικοινωνία μεταξύ αυτών των προγραμμάτων.

Εφαρμογές Java RMI

- Μια εφαρμογή κατανεμημένων αντικειμένων πρέπει να μπορεί να κάνει τα παρακάτω:
  - Εντοπισμό απομακρυσμένων αντικειμένων: Παρέχονται μηχανισμοί αναφοράς σε απομακρυσμένα αντικείμενα. Για παράδειγμα, μια εφαρμογή μπορεί να καταχωρίσει τα απομακρυσμένα αντικείμενα που διατίθενται μέσω του μητρώου RMI, που είναι μια απλή υπηρεσία ονομασίας. Εναλλακτικά μια εφαρμογή μπορεί να περάσει ή να επιστρέψει αναφορές σε απομακρυσμένα αντικείμενα στα πλαίσια μιας κλήσης άλλης απομακρυσμένης μεθόδου.
  - Επικοινωνία με απομακρυσμένα αντικείμενα: Οι λεπτομέρειες της επικοινωνίας με τα απομακρυσμένα αντικείμενα αναλαμβάνονται από το RMI. Για τον προγραμματιστή, η επικοινωνία φαίνεται σαν μια κανονική κλήση μεθόδου.
  - Φόρτωση ορισμών κλάσεων: Το RMI παρέχει μηχανισμούς για το φόρτωμα ορισμών των κλάσεων και των δεδομένων των αντικειμένων που μεταφέρονται.

Αρχιτεκτονική Εφαρμογών Java RMI



Αρχιτεκτονική Εφαρμογών Java RMI

- Πρώτα ο RMI server καλεί το RMI μητρώο (registry) για να καταχωρίσει (bind) ένα απομακρυσμένο αντικείμενο με ένα όνομα.
- Κατόπιν ο RMI client αναζητά το απομακρυσμένο αντικείμενο με το όνομά του στο RMI registry και καλεί μια μέθοδο που εφαρμόζεται στο αντικείμενο.
- Το RMI μπορεί να χρησιμοποιήσει υπάρχοντες web servers στον πελάτη ή και το διακομιστή για να φορτώσει ορισμούς κλάσεων, αν αυτό απαιτείται.

Δυναμική φόρτωση κώδικα

- Ένα από τα κεντρικά και μοναδικά χαρακτηριστικά του RMI είναι η δυνατότητά του να μεταφορτώνει τον ορισμό της κλάσης ενός αντικειμένου αν αυτή η κλάση δεν ορίζεται στην JVM του παραλήπτη.
- Όλοι οι τύποι και οι συμπεριφορές ενός αντικειμένου, όπως ήταν διαθέσιμη σε μια JVM, μπορούν να μεταφερθούν σε μια άλλη JVM.
- Με αυτό τον τρόπο η συμπεριφορά μιας εφαρμογής μπορεί να τροποποιηθεί δυναμικά, εισάγοντας χαρακτηριστικά από άλλο σύστημα.

Interfaces, Κλάσεις και Αντικείμενα

- Μια κατανεμημένη εφαρμογή σε Java RMI κτίζεται με interfaces και κλάσεις.
- Interfaces:
  - ορίζουν μεθόδους.
- Classes:
  - υλοποιούν τις μεθόδους που δηλώθηκαν στα interfaces.
  - πιθανώς ορίζουν και άλλες μεθόδους.
- Objects:
  - ανήκουν σε κλάσεις.
  - σχετίζονται με συγκεκριμένες μεθόδους μέσω συγκεκριμένων interfaces.

Απομακρυσμένα αντικείμενα (1)

- Σε μια κατανεμημένη εφαρμογή, μερικές υλοποιήσεις κλάσεων μπορεί να βρίσκονται σε μια JVM αλλά κάποιες άλλες όχι.
- Τα αντικείμενα που σχετίζονται με μεθόδους που καλούνται από απομακρυσμένες JVMs λέγονται απομακρυσμένα αντικείμενα.
- Ένα αντικείμενο γίνεται απομακρυσμένο μέσω ενός remote interface, με τα παρακάτω χαρακτηριστικά.
  - `extends java.rmi.Remote`.
  - Κάθε μέθοδος της απομακρυσμένης διεπιφάνειας δηλώνει `java.rmi.RemoteException` στη δομή `throws`, πέρα από τις ειδικές εξαιρέσεις της εφαρμογής.

Απομακρυσμένα αντικείμενα (2)

- Το RMI αντιμετωπίζει τα απομακρυσμένα αντικείμενα διαφορετικά από τα τοπικά.
- Αντί να στείλει ένα αντίγραφο της υλοποίησης από τη μια JVM στην άλλη, αυτό που στέλνει είναι ένα στέλεχος (stub) του απομακρυσμένου αντικειμένου.
- Το stub λειτουργεί ώς τοπικός αντιπρόσωπος, ή πληρεξούσιος (proxy), του απομακρυσμένου αντικειμένου.
- Στην ουσία, σε αυτό αναφέρεται το πρόγραμμα πελάτης.

Απομακρυσμένα αντικείμενα (3)

- Ο πελάτης καλεί μια μέθοδο στο proxy, που είναι υπεύθυνο να εκτελέσει την κλήση στο πραγματικό απομακρυσμένο αντικείμενο.
- Το στέλεχος μπορεί να εκτελέσει μόνο τις μεθόδους που ορίζονται στο remote interface και όχι άλλες μεθόδους που πιθανά είναι διαθέσιμες στο απομακρυσμένο αντικείμενο αλλά δεν έχουν δηλωθεί στο remote interface.

02

ΑΝΑΠΤΥΞΗ ΕΦΑΡΜΟΓΗΣ JAVA RMI

Ανάπτυξη Εφαρμογών Java RMI

- Η ανάπτυξη εφαρμογών RMI περιλαμβάνει 4 βήματα:

1. Υλοποίηση των προγραμμάτων client και server. Στην πλευρά του server η εφαρμογή ορίζει το remote interface και υλοποιεί τις κλάσεις των απομακρυσμένων αντικειμένων.
2. Μεταγλώττιση του πηγαίου κώδικα. Αρκεί η χρήση του javac compiler. Πριν την Java 5 έπρεπε να χρησιμοποιηθεί και ο rmic complier.
3. Οι κλάσεις και το remote interface γίνονται γνωστές μέσω διακομιστή ιστού.
4. Εκκίνηση του μητρώου RMI, του server και του client.

RMI Server

- Η εφαρμογή RMI που θα αναπτυχθεί είναι μια απλή 'μηχανή υπολογισμών'.
- Ο RMI server δέχεται από τους πελάτες εργασίες, τις εκτελεί και επιστρέφει τα αποτελέσματα.
- Ο κώδικας του server περιλαμβάνει:
- Την κλάση του απομακρυσμένου αντικειμένου: Παρέχει την υλοποίηση των μεθόδων και των σχετικών αντικειμένων.
- Το remote interface: ορίζει τις μεθόδους του απομακρυσμένου αντικειμένου που μπορεί να καλέσει ο πελάτης.

Απομακρυσμένο interface (1)

- Το **Compute** interface ορίζεται ως απομακρυσμένο μέσω της δήλωσης **extends java.rmi.Remote**.
- Οποιοδήποτε αντικείμενο υλοποιεί αυτό το interface ορίζεται ως απομακρυσμένο αντικείμενο.
- Η μέθοδος **executeTask** είναι η μοναδική μέθοδος-μέλος του remote interface.

```
package compute;

import java.rmi.Remote;
import java.rmi.RemoteException;

public interface Compute extends Remote {
    <T> T executeTask(Task<T> t) throws RemoteException;
}
```

Απομακρυσμένο interface (2)

- Η μέθοδος **executeTask** είναι απομακρυσμένη.
- Δηλώνεται **throws java.rmi.RemoteException**.
- Αυτή η εξαίρεση προκαλείται από το RMI αν κατά την κλήση απομακρυσμένης μεθόδου προκύψει σφάλμα επικοινωνίας.

```
package compute;

import java.rmi.Remote;
import java.rmi.RemoteException;

public interface Compute extends Remote {
    <T> T executeTask(Task<T> t) throws RemoteException;
}
```

Task interface

- Η **executeTask** υλοποιεί το interface **Task**.
- Η **Task** ορίζει ένα γενικευμένο τύπο υπολογιστικής εργασίας που θα εκτελέσει ο server.
- Η **Task** δεν είναι απομακρυσμένη:
  - Δεν υπάρχει δήλωση **extends java.rmi.Remote.**
- Η μοναδική μέθοδος-μέλος είναι η **execute**.

```
package compute;
public interface Task<T> {
    T execute();
}
```

- Η execute δεν έχει παραμέτρους εισόδου.
- Δεν προκαλεί εξαιρέσεις.

Γενικές παρατηρήσεις

- Ένα αντικείμενο που υλοποιεί το **`Compute`** interface μπορεί να εκτελέσει οποιαδήποτε μέθοδο που υλοποιείται μέσω του **`Task`** interface.
- Επειδή τα αντικείμενα που υλοποιούν το interface **`Task`** είναι γραμμένα σε Java μπορούν να φορτωθούν από τη JVM του πελάτη στη JVM του διακομιστή.
- Το RMI μεταφέρει τις τιμές των απομακρυσμένων αντικείμενων (pass by value).
- Χρησιμοποιείται ο μηχανισμός σειριοποίησης αντικειμένων (object serialization) της Java.
- Πρέπει να δηλωθεί με **`implements java.io.Serializable`**.

RMI Server

- Η εφαρμογή server RMI παρέχει αντικείμενα που υλοποιούν το απομακρυσμένο interface (`Compute`).
- Η κλάση των αντικειμένων θα πρέπει να:
  - δηλώνουν το remote interface,
  - ορίζουν ένα constructor για κάθε remote object.
  - υλοποιούν τις απομακρυσμένες μεθόδους των διεπαφών.
- Επίσης, ο server θα πρέπει να:
  - δημιουργήσει τα απομακρυσμένα αντικείμενα,
  - να τα εξάγει στο περιβάλλον εκτέλεσης, και
  - να ενημερώσει το μητρώο RMI, ώστε να μπορούν να δεχτούν απομακρυσμένες κλήσεις.

Υλοποίηση RMI Server

- Η εφαρμογή server RMI παρέχει αντικείμενα που υλοποιούν το απομακρυσμένο interface (`Compute`).
- Η κλάση των αντικειμένων θα πρέπει να:
  - δηλώνουν το remote interface,
  - ορίζουν ένα constructor για κάθε remote object.
  - υλοποιούν τις απομακρυσμένες μεθόδους των διεπαφών.
- Επίσης, ο server θα πρέπει να:
  - δημιουργήσει τα απομακρυσμένα αντικείμενα,
  - να τα εξάγει στο περιβάλλον εκτέλεσης, και
  - να ενημερώσει το μητρώο RMI, ώστε να μπορούν να δεχτούν απομακρυσμένες κλήσεις.

Η κλάση ComputeEngine

```
public class ComputeEngine implements Compute {

public ComputeEngine() { super(); }
public <T> T executeTask(Task<T> t) { return t.execute(); }

public static void main(String[] args) {
        if (System.getSecurityManager() == null) {
            System.setSecurityManager(new SecurityManager());
        }

        try {
            String name = "Compute";
            Compute engine = new ComputeEngine();
            Compute stub = (Compute) UnicastRemoteObject.exportObject(engine,0);

            Registry registry = LocateRegistry.getRegistry();
            registry.rebind(name, stub);
            System.out.println("ComputeEngine bound");
        } catch (Exception e) {
            System.err.println("ComputeEngine exception:");
            e.printStackTrace();
        }
    }
}
```

Ανατομία της `ComputeEngine`

- Υλοποιεί το απομακρυσμένο interface **`Compute`**:
  - Άρα είναι απομακρυσμένο αντικείμενο.
- Έχει υλοποιήσεις για:
  - Constructor (που καλεί τον constructor του **`Compute`** interface),
  - Την απομακρυσμένη μέθοδο **`executeTask`** (έχει οριστεί ως απομακρυσμένη στην υλοποίηση του **`Compute`** interface).
  - Συνάρτηση main() που ξεκινάει ένα αντικείμενο της κλάσης.
- Ο πελάτης θα πρέπει περνάει στην **`executeTask`** ένα αντικείμενο τύπου **`Task`**.
- Ο server θα εκτελεί το **`Task.execute()`** και θα επιστρέφει το αποτέλεσμα στον πελάτη.

Πέρασμα παραμέτρων (1)

- Οι παράμετροι από και προς τις απομακρυσμένες μεθόδους μπορούν να είναι σχεδόν κάθε τύπου:
  - τοπικά & απομακρυσμένα αντικείμενα, βασικοί τύποι δεδομένων...
- Αρκεί να είναι στιγμιότυπο:
  - ενός βασικού τύπου, ή ενός απομακρυσμένου αντικειμένου, ή ενός serializable αντικειμένου (`implements java.io.Serializable`).
- Μη σειριοποιήσιμα αντικείμενα:
  - Threads, file descriptors, και γενικά αντικείμενα που περιέχουν πληροφορία που αφορά συγκεκριμένο χώρο διευθύνσεων.
- Πολλές βασικές κλάσεις όπως τα πακέτα `java.lang` και `java.util`, υλοποιούν το `Serializable` interface.

Πέρασμα παραμέτρων (2)

- Κανόνες περάσματος παραμέτρων:
  - **Τα απομακρυσμένα αντικείμενα** περνούν με αναφορά (by reference). Η αναφορά ενός απομακρυσμένου αντικειμένου είναι ένα proxy stub που στην πλευρά του πελάτη. Υλοποιεί όλα τα απομακρυσμένα interfaces που υλοποιεί το απομακρυσμένο αντικείμενο.
  - **Τα τοπικά αντικείμενα** περνούν με αντιγραφή (by copy), μέσω σειριοποίησης αντικειμένων.
- Pass by reference: οποιεσδήποτε αλλαγές συμβαίνουν στο αντικείμενο, μέσω της κλήσης απομακρυσμένης μεθόδου, εφαρμόζονται στο πρωτότυπο απομακρυσμένο αντικείμενο.

Διαχειριστής ασφάλειας

- Προστατεύει το σύστημα από κακόβουλο κώδικα που πιθανώς να εκτελεστεί στη JVM του διακομιστή.
- Καθορίζει αν ο κώδικας που μεταφορτώθηκε έχει πρόσβαση στο τοπικό σύστημα αρχείων ή μπορεί να εκτελέσει άλλες προνομιακές λειτουργίες.
- Η **`main()`** περιλαμβάνει την εγκατάσταση ενός διαχειριστή ασφάλειας (**`SecurityManager`**).

```
if (System.getSecurityManager() == null) {
        System.setSecurityManager(new SecurityManager());
}
```

Δημιουργία remote αντικειμένου

```
Compute engine = new ComputeEngine();
Compute stub = (Compute) UnicastRemoteObject.exportObject(engine, 0);
```

- Το αντικείμενο αρχικά δημιουργείται από τη **main()**.
- Η μέθοδος **UnicastRemoteObject.exportObject** εξάγει το αντικείμενο ώστε να δέχεται κλήσεις των απομακρυσμένων μεθόδων του από πελάτες.
- Επιστρέφει ένα stub του απομακρυσμένου αντικειμένου με τύπο Compute (όχι ComputeEngine).
- Το δεύτερο όρισμα τύπου int, καθορίζει τη θύρα TCP που θα χρησιμοποιήσει το σύστημα εκτέλεσης RMI για να 'ακούει' τις εισερχόμενες κλήσεις.
  - Συνήθως θέτουμε μηδέν, αφήνοντας την απόφαση στο σύστημα εκτέλεσης του RMI ή στο λειτουργικό σύστημα.

Εγγραφή αντικειμένου στο RMI registry

- Δηλώνεται ένα όνομα για το server.
- Εντοπίζεται το RMI registry (μητρώο).
  - αυτόματα, στην προεπιλεγμένη θύρα 1099 του localhost.
- Το όνομα καταχωρείται στο μητρώο - σε συνδυασμό με το αντίστοιχο στέλεχος του απομακρυσμένου αντικειμένου που έχει δημιουργηθεί.
  - Προσοχή: το stub του αντικειμένου, όχι το αντικείμενο.

```
String name = "Compute";
Registry registry = LocateRegistry.getRegistry();
registry.rebind(name, stub);
```

Εγγραφή αντικειμένου στο RMI registry

- Δεν είναι απαραίτητο να υπάρχει κάποιο είδος βρόχου αναμονής και επεξεργασίας εισερχομένων αιτημάτων.
- Το αντικείμενο **`ComputeEngine`** δεν θα τερματιστεί ούτε θα θεωρηθεί garbage, όσο υπάρχει τουλάχιστο μια αναφορά σε αυτό από κάποια τοπική ή απομακρυσμένη JVM.
- Το αντικείμενο θα τερματιστεί μόνο αν διαγραφεί από το μητρώο RMI και δεν υπάρχουν πελάτες με ενεργές απομακρυσμένες αναφορές σε αυτό.

Πελάτης RMI

- Η υπολογιστική μηχανή είναι απλή: απλώς εκτελεί υπολογιστικές εργασίες που στέλνουν οι πελάτες.
- Μια εφαρμογή πελάτη RMI για την 'υπολογιστική μηχανή' είναι πιο σύνθετη. Περιλαμβάνει:
  - την επικοινωνία με το διακομιστή RMI, αλλά και
  - τον κώδικα της υπολογιστικής εργασίας καθ' αυτής.

Οι δύο κλάσεις του πελάτη RMI

- **`ComputePi`**:
  - αναζητά και αποκτά μια αναφορά στο απομακρυσμένο αντικείμενο **`Compute`**.
  - δημιουργεί μια υπολογιστική εργασία - ένα αντικείμενο **`Task`**,
  - ζητά την εκτέλεση της υπολογιστικής εργασίας από το server μέσω του απομακρυσμένου αντικειμένου.
- **`Pi`**:
  - υλοποιεί το **`Task`** interface.
  - περιγράφει την υπολογιστική εργασία:
    - δηλ. τον υπολογισμό του π με δεδομένη ακρίβεια δεκαδικών ψηφίων.
  - έχει ως μοναδικό όρισμα την απαιτούμενη ακρίβεια και επιστρέφει ένα **`java.math.BigDecimal`**.

Η κλάση ComputePi (1)

```
package client;
import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;
import java.math.BigDecimal;
import compute.Compute;

public class ComputePi {
     public static void main(String args[]) {
          if (System.getSecurityManager() == null) {
               System.setSecurityManager(new SecurityManager());
          }
          try {
               String name = "Compute";
               Registry registry = LocateRegistry.getRegistry(args[0]);
               Compute comp = (Compute) registry.lookup(name);
               Pi task = new Pi(Integer.parseInt(args[1]));
               BigDecimal pi = comp.executeTask(task);
               System.out.println(pi);
          } catch (Exception e) {
               System.err.println("ComputePi exception:");
               e.printStackTrace();
          }
     }
}
```

Η κλάση `ComputePi` (2)

- Όπως και ο διακομιστής, έτσι και ο πελάτης ξεκινά με την εγκατάσταση ενός διαχειριστή ασφάλειας.
- Το σύστημα εκτέλεσης RMI δεν θα επιτρέψει τη μεταφόρτωση αν δεν λειτουργεί διαχειριστής ασφάλειας.
- Ορίζεται το όνομα του απομακρυσμένου αντικειμένου. Το όνομα, **`Compute`**, πρέπει να είναι το ίδιο με αυτό που χρησιμοποίησε ο διακομιστής για να καταχωρήσει το αντικείμενο στο μητρώο RMI.
- Πρέπει να είναι γνωστό εκ των προτέρων.

Η κλάση `ComputePi` (3)

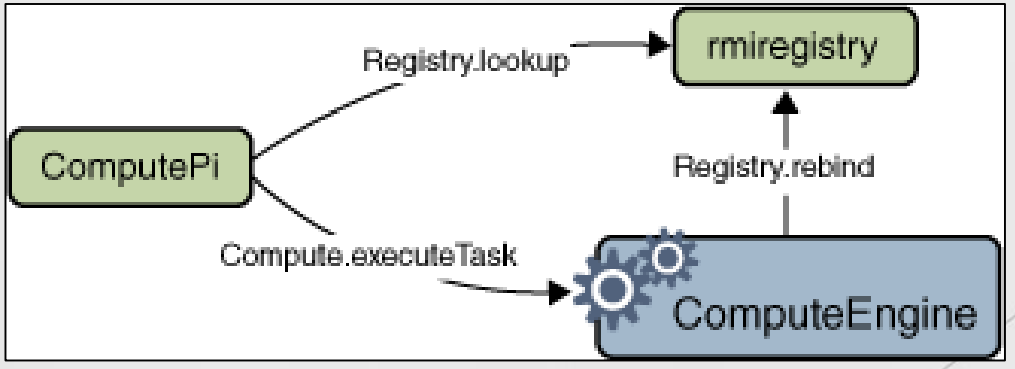
- Ο πελάτης αναζητά το μητρώο RMI του server.
- Η **`LocateRegistry.getRegistry`** επιστρέφει μια αναφορά στο μητρώο RMI του server.
- Πρέπει να είναι γνωστή εκ των προτέρων.
- Η **`LocateRegistry.getRegistry`** μπορεί να κληθεί και με δύο ορίσματα, το δεύτερο τύπου **`int`**, σε περίπτωση που το μητρώο RMI δεν 'ακούει' στην προεπιλεγμένη θύρα 1099 αλλά σε κάποιά άλλη.
- Ο client αποκτά μια αναφορά στο απομακρυσμένο αντικείμενο μέσω της μεθόδου **`registry.lookup`**.
- Η **`registry.lookup`** δέχεται ως όρισμα το όνομα του αντικειμένου και επιστρέφει ένα αντίγραφο **`comp`** του στελέχους του απομακρυσμένου αντικειμένου που υλοποιεί το **`Compute`** interface.

Η κλάση `ComputePi` (4)

- Στη συνέχεια ο πελάτης δημιουργεί την υπολογιστική εργασία, δηλαδή ένα αντικείμενο Pi task που υλοποιεί το **`Task`** interface.
- Ο πελάτης καλεί την απομακρυσμένη μέθοδο **`executeTask`** επί του απομακρυσμένου αντικειμένου **`Compute comp`**.
- Η **`executeTask`** δέχεται ως παράμετρο την εργασία **`task`**, η οποία αφού εκτελεστεί στο server επιστρέφει ένα αντικείμενο τύπου **`BigDecimal`**, το οποίο αποθηκεύεται στο result.
- Ο πελάτης πριν τερματίσει εμφανίζει το αποτέλεσμα.

Το σύστημα επικοινωνίας RMI

- Επικοινωνία μεταξύ του πελάτη **`ComputePi`**, του μητρώου **`rmiregistry`**, και του διακομιστή **`ComputeEngine`**.



Η κλάση Pi (1)

```
package client;
import compute.Task;
import java.io.Serializable;
import java.math.BigDecimal;

public class Pi implements Task<BigDecimal>, Serializable {
	private static final long serialVersionUID = 227L;
	private static final BigDecimal FOUR = BigDecimal.valueOf(4);
	private static final int roundingMode = BigDecimal.ROUND_HALF_EVEN;
	private final int digits;
	public Pi(int digits) { this.digits = digits; }
	public BigDecimal execute() { return computePi(digits); }

	public static BigDecimal computePi(int digits) {
		int scale = digits + 5;
		BigDecimal arctan1_5 = arctan(5, scale);
		BigDecimal arctan1_239 = arctan(239, scale);
		BigDecimal pi = arctan1_5.multiply(FOUR).subtract(arctan1_239).multiply(FOUR);
		return pi.setScale(digits, BigDecimal.ROUND_HALF_UP);
	}

	public static BigDecimal arctan(int inverseX, int scale) {
		BigDecimal result, numer, term, invX = BigDecimal.valueOf(inverseX);
		BigDecimal invX2 = BigDecimal.valueOf(inverseX * inverseX);

		numer = BigDecimal.ONE.divide(invX, scale, roundingMode);
		result = numer;
		int i = 1;
		do {
			numer = numer.divide(invX2, scale, roundingMode);
			int denom = 2 * i + 1;
			term = numer.divide(BigDecimal.valueOf(denom), scale, roundingMode);
			if ((i % 2) != 0) {
				result = result.subtract(term);
			} else {
				result = result.add(term);
			}
			i++;
		} while (term.compareTo(BigDecimal.ZERO) != 0);
		return result;
	}
}
```

Η κλάση `Pi` (2)

- Η κλάση **`Pi`** υλοποιεί το **`Task`** interface.
- Υπολογίζει το π για δοσμένη ακρίβεια.
- Σειριοποιήσιμες κλάσεις: κλάσεις που υλοποιούν τη διεπιφάνεια **`Serializable`** έμμεσα ή άμεσα.
  - πρέπει να δηλώσουν ένα πεδίο private static final με όνομα serialVersionUID που διασφαλίζει τη συμβατότητα μεταξύ διαφορετικών σειριοποιημένων εκδόσεων.
- Αν δεν υπάρχει προηγούμενη έκδοση της κλάσης, τότε η τιμή του πεδίου αυτού μπορεί να είναι οποιαδήποτε τιμή **`long`** value, όπως η **`227L`**.
- Αρκεί αυτή η τιμή να διατηρηθεί και σε μελλοντικές εκδόσεις.

Η κλάση `Pi` (3)

- Η ουσιαστική εκτέλεση της εφαρμογής ξεκινά όταν το αντικείμενο **`Pi task`** περναέι ως παράμετρος στη μέθοδο **`executeTask`**.
- Ο κώδικας της κλάσης **`Pi`** μεταφορτώνεται στην JVM του server ως υλοποίηση του interface **`Task`**.
- Η μέθοδος **`comp.executeTask(task)`** του client εκτελείται ως **`task.execute()`** στο server.
- Η μέθοδος **`task.execute()`** χρησιμοποιεί για την εργασία τη μέθοδο **`task.computePi(digits)`**.
- Το αποτέλεσμα είναι το αντικείμενο **`BigDecimal`**, που επιστρέφεται στο πελάτη ως **`pi`**, το οποίο και τυπώνεται ως αποτέλεσμα του υπολογισμού.

Java packages

- Η υλοποίηση περιλαμβάνει 3 Java packages:
  - compute – **`Compute`** και **`Task`** interfaces.
  - engine – Κλάση **`ComputeEngine`** (server).
  - client – κλάσεις **`Pi`** και **`ComputePi`**.
- Διαχείριση πακέτων ως αρχείων JAR.
- Τυπικά, ένας προγραμματιστής θα ετοιμάσει τα δύο πρώτα πακέτα.
- Τα interfaces θα πρέπει να είναι διαθέσιμα ώστε κάποιοι άλλοι προγραμματιστές να τις μεταφορτώσουν και να ετοιμάσουν το τρίτο πακέτο, δηλαδή τους πελάτες.

03

COMPILATION
ΕΦΑΡΜΟΓΗΣ

Compilation (interfaces - compute)

- Compilation των interfaces (**compute** package – **Task** & **Compute** κλάσεις) & δημιουργία του **compute.jar**:

```
cd C:\Users\leo\eclipse-workspace\RMI\src\compute
D:\Progra~1\Java\jdk1.8.0_144\bin\javac Compute.java Task.java
D:\Progra~1\Java\jdk1.8.0_144\bin\jar cvf compute.jar *.class
```

- Το αρχείο **compute.jar** μπορεί να διανεμηθεί στους προγραμματιστές που θα γράψουν τον κώδικα εφαρμογής πελάτη.
- Η JVM του πελάτη πρέπει να έχει αυτές τις κλάσεις στο class path της.
- Το ίδιο και το RMI registry.

Διάθεση κλάσεων ή αρχείων JAR

- Απλός τρόπος διάθεσης αρχείων class ή JAR: τοποθέτησή τους σε ένα public κατάλογο Web.
- Απαιτείται τοπικός ή δημόσιος Web server.
- Π.χ. WampServer - https://www.wampserver.com/en/
- WAMP Public directory: **`www`**
- Στο παράδειγμα μας: **`D:/wamp/www/www_tech`**.
  - Ορατός ως: http://localhost/www_tech.
- Εκεί τοποθετούμε το αρχείο compute.jar.

Compilation (server – engine package)

- Compilation κλάσης server (**engine** package – κλάση **ComputeEngine**) & δημιουργία του **engine.jar**.
- Απαιτείται το **compute.jar** στο class path.
  - Δηλαδή τα δύο interfaces **Compute** και **Task**.
  - Το τοποθετήσαμε στο public directory του Web server.

```
D:\Progra~1\Java\jdk1.8.0_144\bin\javac
        -cp D:\wamp\www\www_tech\compute.jar
        engine\ComputeEngine.java
```

- Interfaces JAR στο public Web directory.
- Target class (**ComputeEngine**).

Compilation (client – `client` package)

- Compilation κλάσεων client (client package – κλάσεις **`Pi`** και **`ComputePi`**) & δημιουργία του **`client.jar`**.
- Πάλι απαιτείται το **`compute.jar`** στο class path.
  - Δηλαδή τα δύο interfaces **`Compute`** και **`Task`**.
  - Το τοποθετήσαμε στο public directory του Web server.

```
D:\Progra~1\Java\jdk1.8.0_144\bin\javac
        -cp D:\wamp\www\www_tech\compute.jar
        client/ComputePi.java client/Pi.java
```

- Interfaces JAR στο public Web directory.
- Target classes (**`Pi`** και **`ComputePi`**).
- Μόνο η κλάση **`Pi`** δημοσιοποιείται στο Web server.

04

ΕΚΤΕΛΕΣΗ ΕΦΑΡΜΟΓΗΣ

Εκκίνηση του Μητρώου RMI

- Πριν εκτελέσουμε το πρόγραμμα του server πρέπει να ξεκινήσουμε το RMI registry στο σύστημα του server.

```
D:\Progra~1\Java\jre1.8.0_144\bin\rmiregistry.exe
```

- Εξ' ορισμού το μητρώο 'ακούει' στη θύρα 1099. Μπορούμε να ορίσουμε διαφορετική θύρα ως εξής:

```
D:\Progra~1\Java\jre1.8.0_144\bin\rmiregistry.exe 2001
```

Εκκίνηση του Server

- Στην εντολή εκκίνησης πρέπει να ορίσουμε τα εξής:
- Τα class paths που περιλαμβάνουν τα εκτελέσιμα του server και των interfaces.
- Το URL όπου δημοσιοποιούνται τα αρχεία του server.
  - Παράμετρος **`java.rmi.server.codebase`**
- Το όνομα του server. Το σύστημα εκτέλεσης του RMI χρησιμοποιεί by default την διεύθυνση IP που επιστρέφει **`java.net.InetAddress.getLocalHost`**.
  - Με το **`java.rmi.server.hostname`** ορίζουμε εναλλακτική IP.
- Το αρχείο πολιτικών ασφαλείας.
  - Παράμετρος **`java.java.scurity.policy`**.

Εκκίνηση του Client

- Στην εντολή εκκίνησης πρέπει να ορίσουμε τα εξής:
- Τα class paths που περιλαμβάνουν τα εκτελέσιμα του server και των interfaces.
- Το URL όπου δημοσιοποιούνται τα αρχεία του client.
  - Παράμετρος java.rmi.server.codebase.
- Το αρχείο πολιτικών ασφαλείας.
  - Παράμετρος **`java.java.scurity.policy`**.
- Περνούμε τα δύο ορίσματα της γραμμής εντολών του πελάτη, το όνομα του διακομιστή και την ακρίβεια υπολογισμού του pi symbol.