Lab 1 - Java RMI
Remote Method Invocation

Object-oriented RPC (Remote Procedure Call)

Allows calling methods of a remote object (over a network) through standard invocation of interface methods

Guide
https://docs.oracle.com/javase/8/docs/technotes/guides/rmi/index.html

API
https://docs.oracle.com/javase/8/docs/api/java/rmi/package-summary.html

Spec
https://docs.oracle.com/javase/8/docs/platform/rmi/spec/rmiTOC.html
Hello World

• Shows how use RMI to:
  ■ run a server which
    ▪ creates an object that accepts remote calls
    ▪ registers this object under a name in a registry
  ■ run a client which
    ▪ retrieves a proxy to a remote object from a registry
    ▪ calls methods of the remote object on the server
    ▪ passes values to and from the remote object

• Download example and assignment files
  https://d3s.mff.cuni.cz/legacy/files/teaching/nswi080/labs/F1.zip
Java Sources

- **Example.java**
  - remote interface

- **ExampleImpl.java**
  - implementation of remote interface
  - instances are remote objects

- **ExampleServer.java**
  - creates a remote object
  - registers the object in a registry
  - accepts calls to the remote object

- **ExampleClient.java**
  - retrieves a proxy to the remote object
  - calls a method of a remote object
Scripts

- **make**
  - run by `bash make`
  - compile java files using `javac`
  - generating stub classes using `rmic` is not done anymore, they are generated dynamically

- **run-registry**
  - starts a `rmiregistry` process
  - run "in the background"
  - Port in use? – use different port number ($> 1024$)
    - Edit path in calls to `[re]bind()` and `lookup()`
    - `localhost` becomes `localhost:1234`
  - for simplicity, all scripts use the same classpath
    - we avoid setting up codebase, permissions, ...
Scripts

- run-server
  - starts the server
  - kill by Ctrl+C
  - `java.rmi.server.hostname` = what address will client use to call methods on the server (address stored in proxy objects)

- run-client
  - runs the client
Task 1 - Java RMI
public interface Searcher {
    public int getDistance(Node from, Node to);
}
public interface Node {
    Set<Node> getNeighbors();
    void addNeighbor(Node neighbor);
}
Node[] graph;
1 - Local implementation

- Interfaces `Node` and `Searcher`
- Breadth-first graph search algorithm
- Local implementation provided
- Executable `Main` class
  - generates a random graph
  - measures speed of random queries
Task

- Extend the provided implementation to search the graph remotely
- Compare speed of various configurations
2 - Remote Searcher

- Extend the **Searcher** interface (see Example)
  - Interface `java.rmi.Remote`
  - Exception of type `java.rmi.RemoteException`

- Remotely accessible object (see ExampleImpl)
  - Must be exported – 2 possible ways
  - Derive from `java.rmi.server.UnicastRemoteObject`
    - Export ensured by parent constructor
  - Call `UnicastRemoteObject.exportObject(obj)` manually
    - Does not handle semantics of `hashCode()`, `equals()`, `toString()` – not a problem with Searcher (just one instance)
3 - Remote Node Objects

- Extend interface `Node` with RMI (like `Searcher`)

- Class inherits from `UnicastRemoteObject` and implements `Node`
  - To allow for `hashCode()`, `equals()`, `toString()`
  - Copy/paste + edit is enough
    - We want the local `Node` s in the previous part still behave locally
3 - Remote Node Objects (cont.)

- How to create and return instances for client requests?

- Implement `NodeFactory` with a method `createNode()`
  - Similar to remote `Searcher` – an interface with RMI, an implementing class, create an instance and call `Naming.bind()` inside the existing server
  - Do not create a separate server for nodes, we want just one for the next variant

- Client gets the reference using `lookup()`, then creates the remote `Node` objects together with the local graph

- How does the local `Searcher` access the remote `Node`s?

- What exactly does the `NodeFactory` return to the client?
Everything is ready, just add this variant to `searchBenchmark()` and compare the speed.

How does the `Searcher` on server access the `Node` objects on (the same) server?
Impact of the Network

- So far, client and server were running on the same machine
  - Overhead of RMI communication, but no network latency

- Run on more machines
  - Server on the machine next to you, client on yours
  - Change the paths in `rebind()` and `lookup()`
    - Remote machine name instead of `localhost`
    - Modify `main` to use `args[0]`
  - Run `rmiregistry` and `Server` in an SSH session on the remote machine
  - Run the client locally
    - Beware of `CLASSPATH`
Passing by Value vs. Reference

- Previous tasks show “extreme” cases
- How about combining both approaches?
- Idea: “batch” transfer bigger parts of the graph

- `getTransitiveNeighbors(int distance)`
  - Returns all neighbors up to some distance

- Use the `getDistanceTransitive` method of the `Searcher` interface
  - In each step, requests neighbors up to the specified distance

- Choose a reasonable value for the distance parameter
  - Compare measured times with previous variants
Extend the single project, do not create 4 separate ones.

- Interface hides different implementations
  - Even `Remote` interface can be used locally
    - Just catch exceptions that would never occur
  - E.g. remote graph is just another array `Node[]`
    - Easy to have the same (logically) local and remote one
    - Similarly with `Searcher`

- Measure everything in one run to ease comparison
  - Just add measuring and a column to results in `searchBenchmark()`
Submission

- Working implementation
- Documentation
  - Answer all the questions from the assignment
  - Describe measurement results
- By e-mail (deadline is on the web)
- Make sure it works in the lab downstairs
- The submission shall be easy to start
  - Use the provided implementation
  - No need for Maven or Ant script
  - Do not add packages etc.
  - Updated versions of the `run-server` scripts