Connecting to the server

Socket socket = new Socket(); - getting socket  
socket.connect(new InetSocketAddress("address", port), timeout); - connecting to the server with special port and with timeout if there will be error  
Scanner scanner = new Scanner(socket.getInputStream()); - getting reader from the data

Creating server

try(ServerSocket serverSocket = new ServerSocket(8179)) { - create serversocker on the 8179 port  
  
 Socket socket = serverSocket.accept(); - wait for the request  
 Scanner scanner = new Scanner(socket.getInputStream()); - creating inputstream (place, where they will be exchanging files and info)

PrintWriter writer = new PrintWriter(socket.getOutputStream(), true); - creating writer  
 writer.println("qwerty"); - writing into the stream

**HTTP connection**

URLConnection connection = new URL("http://google.com").openConnection(); - connect to the site  
Scanner scanner = new Scanner(connection.getInputStream()); - get stream   
scanner.useDelimiter("\\Z"); - don’t know, needed for correct values in next method  
  
System.out.println(scanner.next()); - get html  
  
Map<String, List<String>> parameters = connection.getHeaderFields(); - get all parameters of connection   
for (Map.Entry<String, List<String>> p : parameters.entrySet()){ - turn into set  
 System.out.println(p.getKey() + " / " + p.getValue()); - getting values  
}

**Create own server, which will send variable of class and another program will use its methods**

In this way: **…\IdeaProjects\proj\out\production**\**proj** set this command **rmiregistry**

And set this VM option: **-Djava.rmi.server.codebase=http://localhost:**1099**/**fuck

public class server { - another file with server  
 public static void main(String[] args) throws NamingException {

Context context = new InitialContext(); - create context  
 context.rebind("rmi://localhost:1099/fuck", new Fuck()); - in main function rebind socket and set your class variable there  
 }  
}  
class Fuck extends UnicastRemoteObject implements example{

@override  
 public int add(int a, int b) throws RemoteException{  
 return a + b;  
 }  
 Fuck() throws RemoteException{ - needed function   
 }  
}

public interface exampl extends Remote { - in another file create this interface and extend Remote  
 public int add(int a, int b) throws RemoteException;  
}

static public void start() throws Exception{ - in your main program (car for example )  
  
 Context context = new InitialContext();  
 Enumeration<NameClassPair> e = context.list("rmi://localhost");  
 while (e.hasMoreElements()){  
 e.nextElement().getName(); - get name of the class  
 exampl func = (exampl)context.lookup("rmi://localhost/fuck"); - get class by transforming into interface and get from method, which connects you with another computer  
 int qwe = func.add(3,6); - example of using functions   
 }  
  
  
}

**MYSQL**

Statement state = conn.createStatement();

ResultSet result = state.executeQuery("select \* from test1");

Result.getDate(“from where”) - get date

**Connect to the DB**

String url = "jdbc:mysql://localhost:3306/world?useLegacyDatetimeCode=false&serverTimezone=Australia/Melbourne&useSSL=false"; - url of the db, first part is located at the instructions of the db, second just copy, it is important   
Class.forName("com.mysql.cj.jdbc.Driver"); - set driver, it is important  
  
try(Connection conn = DriverManager.getConnection(url, username, pass)){…} - connecting

Statement statement = conn.createStatement(); - creating management  
statement.executeUpdate(" "); - create/delete/update  
  
ResultSet rs = statement.executeQuery("select \* from world"); - select info  
while (rs.next()){ - if ha next and get next  
 System.out.println(rs.toString()); - get str  
}

Protect from sql injection

PreparedStatement statement = conn.prepareStatement("select \* from city where id = ?"); - prepare request with ? instead of pasted value  
statement.setString(1,"1"); - set value by key and value (can also setInt, setBolean …)  
ResultSet result = statement.executeQuery(); - get result

Record images

Record

BufferedImage image = ImageIO.read(new File("…")); - get image  
Blob blob = connection.createBlob(); - create blob, connected with database  
try(OutputStream outputStream = blob.setBinaryStream(1)) { - get outputStream  
 ImageIO.write(image, "jpg", outputStream); - record image to the outputStream of the blob  
}  
PreparedStatement statement = conn.prepareStatement("insert into test1 (name, image) values (‘…’, ?)"); - create request with undefined property  
statement.setBlob(1, blob); - set undefined property  
statement.execute(); - record

read

Statement state = conn.createStatement(); - create connection variable   
ResultSet result = state.executeQuery("select \* from …"); - get table  
while (result.next()){ - run  
 Blob blob1 = result.getBlob("image"); - get field  
 BufferedImage image1 = ImageIO.read(blob.getBinaryStream()); - record to the image  
 ImageIO.write(image1,"png",new File("pisos")); - save file  
}

**Iterate through the results**

Statement state = conn.createStatement(ResultSet.TYPE\_SCROLL\_INSENSITIVE, ResultSet.CONCUR\_READ\_ONLY); - get state, which is able to get different methods, not only next ( TYPE\_SCROLL\_INSENSITIVE – enable to iterate without looking at changing elements TYPE\_SCROLL\_SENSITIVE – inversion, CONCUR\_READ\_ONLY – can’t change)  
  
ResultSet set = state.executeQuery("select \* from city"); - get results  
if (set.next()){…} – returns if it exists next and if it is, iterator goes to him  
set.previous() – returns if it exists previous and if it is, iterator goes to him  
set.relative(…) – returns if it exists element number … from the current and if it is, iterator goes to him  
set.absolute(…) – returns if it exists element number … from the start and if it is, iterator goes to him  
set.first() – returns first  
set.last() – returns last  
  
set.beforeFirst(); - set element before first (needed if you are going to iterate all elements using function next() )  
  
set.afterLast(); - set element after last (needed if you are going to iterate all elements using function previous() )

**Insert, update, delete**

Statement state = conn.createStatement(ResultSet.TYPE\_SCROLL\_SENSITIVE, ResultSet.CONCUR\_UPDATABLE); - get state

State. executeQuery(“…”) – command to the database  
ResultSet set = state.executeQuery("select \* from city"); - get result  
  
set.updateString("column name", "value"); - update line  
set.updateRow(); - insert line  
  
set.moveToInsertRow(); - move to new line  
set.updateString("column name", "value"); - create new line  
set.insertRow(); - insert line  
  
set.deleteRow(); - delete current line

Get info about tables

Statement statement = conn.createStatement(); - get statement   
ResultSet set = statement.executeQuery("select \* from city"); - get results  
ResultSetMetaData metaData = set.getMetaData(); - get info about table  
for (int i = 1; i <= metaData.getColumnCount(); i++) { - run though the columns  
 System.out.println(metaData.getColumnLabel(i) + " - " + metaData.getColumnTypeName(i)); - get info about table for example name of the label and its type  
}

**Use database in another method with this connection**

{  
 RowSetFactory factory = RowSetProvider.newFactory();  
 CachedRowSet set1 = factory.createCachedRowSet();  
 Statement state = conn.createStatement(ResultSet.TYPE\_SCROLL\_SENSITIVE, ResultSet.CONCUR\_UPDATABLE);  
 ResultSet set = state.executeQuery("select \* from city");  
 set1.populate(set);  
 return set1;  
} – another method, which returns CachedRowSet of the database, class, which can transfer access to database through different methods, if you return only ResultSet, it wont work

{

CachedRowSet set = (CachedRowSet) Myclass.start(); - get variable  
set.setCommand("select \* from city where id = 1"); - if there are lots of variables, choose what you need  
set.setPageSize(10); - max size  
set.execute(DriverManager.getConnection(url, username, pass)); - set the settings like 2 on the top  
set.setTableName("…"); - get table from the database, with which you will work  
set.acceptChanges(DriverManager.getConnection(url, username, pass)); - if you changed something, this method will add this changes to database

} in other methods like update, get, delete… class CachedRowSet is similar to the ResultSet

Transactions

Functions update database only after all operations,

Used to avoid losing data if there is exception

conn.setAutoCommit(false); - commin only when you use method conn.commit();

conn.commit(); - update all data

Savepoint savepoint = conn.setSavepoint(); - set here savepoint means you will save all info on the top from the rollback  
conn.rollback(savepoint); - delete all updates till the savepoint, mostly used in catching exceptions   
conn.releaseSavepoint(savepoint); - if you used rollback, it will reconstruct saved information (upper than setting point)

https://www.youtube.com/watch?v=5Z2iFX3OeTo&list=PL786bPIlqEjRDXpAKYbzpdTaOYsWyjtCX&index=202

conn.setTransactionIsolation(Connection.TRANSACTION\_NONE); - sets isolation rule, rarly needed for example if you work with one db in different threads

Faster way to use database (records at once instead of connecting each tine to db )

conn.setAutoCommit(false); - commin only when you use method conn.commit();  
Statement statement = conn.createStatement(); - get statement  
statement.addBatch(); - create new request   
statement.executeBatch().length; - returns array with number of batches  
conn.commit(); - commit all batches at once