Useful stuff in Java 7

New stuff that's actually useful

Try with resources

Recognize this?

```
try {
  stmt = con.createStatement();
  ResultSet rs = stmt.executeQuery(query);
  System.out.println("Coffees bought from " + supplierName + ": ");
  while (rs.next()) {
    String coffeeName = rs.getString(1);
    System.out.println("     " + coffeeName);
  }
} catch (SQLException e) {
  JDBCTutorialUtilities.printSQLException(e);
} finally {
  if (stmt != null) { stmt.close(); } // close() may throw an exception!
}
```

Great if resources could close themselves

```
try (Statement stmt = con.createStatement()) {
  ResultSet rs = stmt.executeQuery(query);
  while (rs.next()) {
    String coffeeName = rs.getString(1);
    System.out.println("     " + coffeeName);
  }
} catch (SQLException e) {
  JDBCTutorialUtilities.printSQLException(e);
}


// stmt is automatically closed in an implicit finally clause

// Note, however, if close() throws an exception it will be surpressed
```

Is it magic?

No. It’s a new interface:

```
public interface java.lang.AutoCloseable {
  public abstract void close() throws java.lang.Exception;
}
```

Catching more than one (related) exception

Consider this:

```
try{
  // Some code that might throw multiple exceptions
}catch(FileNotFoundException fnfe){
  // Handle this - couldn't read file
}catch(AccessDeniedException ade){
  // Handle this - couldn't read file
}
```

Catching more than one (related) exception

If the handling of the two cases are the same (let’s pretend it is), then you can write like this since Java 7:

```
try{
  // Some code that might throw multiple exceptions
}catch(FileNotFoundException|AccessDeniedException e){
  // Handle this - couldn’t read file
}

(Exceptions that are related via inheritance doesn’t need this, you could
 then instead catch the super class)
```

Path is the new File!

A path is an abstract representation of a path in the file system. Some useful ways for creating a Path instance (Path is an interface):

```
Path p         = Paths.get("/", "home", "rikard", "yrgo", "file.txt");
Path home      = Paths.get("/home/rikard");
Path yrgo      = home.resolve("yrgo");             //  /home/rikard/yrgo
Path chalmers = yrgo.resolveSibling("chalmers"); //  /home/rikard/chalmers
Path fromYrgoToChalmers
               = yrgo.relativize(chalmers);        //  ../chalmers
// See also: getParent(), getFileName(), getRoot()
```

Using Paths to read a file with text

```
Path p  = Paths.get("/home/rikard/yrgo/file.txt");
String fileContents =
   new String(Files.readAllBytes(p), StandardCharsets.UTF_8);

// Line-by-line with line numbers:
List<String> lines = Files.readAllLines(p, StandardCharsets.UTF_8);
int lineNumber=1;
for(String line : lines){
  System.out.println(lineNumber++ + line);
}
```

Let’s write a text file!

```
Path targetFile = Paths.get("/home/rikard/yrgo/my_target.txt");
String content = "Första raden.\nAndra raden.\nTredje raden.";
Files.write(targetFile, content.getBytes(StandardCharsets.UTF_8));

// Append two more lines:
String extra = "\nFjärde raden.\nFemte raden.\n";
Files.write(targetFile, extra.getBytes(StandardCharsets.UTF_8),
            StandardOpenOption.APPEND);

// Append a list of lines:
List<String> lastLines = new ArrayList<String>();
lastLines.add("Sjätte raden.");lastLines.add("Sjunde raden.");
Files.write(targetFile, lastLines, StandardCharsets.UTF_8,
            StandardOpenOption.APPEND);
```

More stuff in Files

Remember this?

```
BufferedReader br = new BufferedReader(new FileReader(fileName));
PrintWriter out   = new java.io.PrintWriter(
                       new java.io.BufferedWriter(
                        new java.io.FileWriter(filename + ".java")));
```

Now you can do this:

```
InputStream in   = Files.newInputStream(path);
OutputStream out = Files.newOutputStream(path);
Reader reader    = Files.newBufferedReader(path);
Writer writer    = Files.newBufferedWriter(path);
```

Copying streams to files and vice versa

```
Files.copy(in, path);

Files.copy(path, out);
```

Copying, moving, deleting files, mkdir, touch?

```
Files.copy(fromPath, toPath);
Files.move(fromPath, toPath);
Files.delete(path);
boolean deleted = Files.deleteIfExists(path);

Files.createDirectory(path);
Files.createDirectories(path); // same as mkdir -p many/dirs/in/one/go

Files.createFile(path); // create new empty file
```

Implementing LS using DirectoryStream

```
String dirName = args[0];
Path path = Paths.get(dirName);
try (DirectoryStream<Path> dir =
     Files.newDirectoryStream(path, "*.java")) // filter on *.java
{
  for (Path file : dir){
    System.out.println(file.getFileName());
  }
}catch(Exception e){}
```

Null safe equals?

```
if(s!=null && s.equals(QUIT_COMMAND)){
  System.exit(0);
}

// Using Objects.equals:
if(Objects.equals(s, QUIT_COMMAND)){
  System.out.println("Bye");
  System.exit(0);
}
```

Diamond league - the ‹› operator

This might give you a headache:

```
Map<String, List<Integer>> map = new HashMap<String, List<Integer>>();
```

Since Java7 you may instead write:

```
Map<String, List<Integer>> map = new HashMap<>();

// Think “yada, yada, yada” ;-)
```

New formats for literals

```
int salary = 38_000;

int studentLoan = 1_300_000;
```