Programming in JAVA

lecture 10

Exceptions, file input/output

Exceptions

1. In JAVA, Exceptions are used to deal with errors that occur during code executions.
2. Some methods in library classes are designed in such a way that when using them, programmers have to intercept exceptions.
3. To intercept exception, we use try..catch keywords. tables, but more convenient in business applications.
4. Optionally, we can use the finally keyword to specify operation that should execute in both situations (when the is error or the operations completed successfully).

The try..catch

```
try {
   // operation that can throw exceptions
} catch (Exception e)
{
   System.out.println("error 3244");
}
finally
{
  // this block will execute always, no matter if
  // an exception has been thrown or not
}
```

Exceptions with file I/O

1. Dealing with exceptions is mandatory when programming file Input/Output operations.
2. Exceptions could be throwns both in file opening, and in file operations.
3. The cause of exception throwing could be :
   - program do not have permission to write to location it is trying,
   - disk is full and no additional data can be written,
   - the catalog the program is writting has been removed from filesystem (e. g. the user removed a pendrive)

File I/O classes

1. To read from a file we use the `FileInputStream` class.
2. To write to a file we use the `FileOutputStream` class.
3. First we open a file by creating a new object of the file stream class. As an argument we put the path and the name of the file.
4. Then we use method from file stream class to write/read from a file.
5. Finally we close the file.
6. In case of reading from a file, we detect the end of file when the method `read` retururns value -1.

Reading from a file

```
int tmp;
FileInputStream in=null;
try {
      in = new FileInputStream("abc.txt");
      while (  (tmp=in.read())!=-1){
      System.out.print((char)tmp);
}

catch(FileNotFoundException e) {
     e.printStackTrace();
}catch(IOException e) {
     e.printStackTrace();
}finally {
      try{
        if (in!=null) in.close();
      }catch(IOException e) {
        e.printStackTrace();
  }
}
```

Reading from a file by blocks (only first part of code)

```
byte[] buffer = new byte[10];
FileInputStream in=null;


try {
      in = new FileInputStream("abc.txt");
      while( (tmp=in.read(buffer))==buffer.length ) {
          System.out.print(new String(buffer));
      }

     for(int i=tmp;i<buffer.length ;i++)
            buffer[i]=0;
     System.out.println(new String(buffer));
```

Writing to a file

```
String str = "Something to write into a file";
FileOutputStream out=null;

try {
     out = new FileOutputStream("abc2.txt");
     // writing byte after byte
     for(int i=0; i< str.length() ; i++)
             out.write(str.charAt(i));

     // writing entire block (buffer)
     //out.write(str.getBytes());
```

Copying files

```
Scanner sc = new Scanner(System.in);
System.out.print("enter file you want to copy:");
String file_src = sc.nextLine();
System.out.print("enter name of destination file :");
String file_dsc = sc.nextLine();

FileInputStream in=null;
FileOutputStream out=null;
```

Coping file (only key lines of code)

```
try {
       in = new FileInputStream(file_src);
       out = new FileOutputStream(file_dsc);
       int tmp;
       while( (tmp=in.read())!=-1 )
            out.write(tmp);

}
```

Coping file in block mode (only key lines of code)

```
byte[] buffer = new byte[10];

try {
      in = new FileInputStream(file_src);
      out = new FileOutputStream(file_dsc);
      int tmp;
      while( (tmp=in.read(buffer))!= -1){
        out.write(buffer,0,tmp);
      }
} catch ….
```