Programowanie obiektowe

Wykład 10: Strumienie we/wy

Zarządzanie systemem plików

klasa File (pakiet java.io)

np. **File path=new File(".");**

Metody: **list(), getName(), delete(), exists(), isDirectory(), isFile(), mkdir(), renameTo(), length(), lastModified(), canRead(), canWrite(), …**

Klasy podstawowe

- InputStream
  - ByteArrayInputStream(tablica bajtów)
  - StringBufferInputStream(String)
  - FileInputStream(String lub File)
  - PipedInputStream(PipedOutput Stream)
  - SequenceInputStream(kontener strumieni wejściowych lub 2 strumienie wejściowe)
- OutputStream
  - ByteArrayOutputStream(początkowa wielkość bufora)
  - FileOutputStream(String lub File)
  - PipedOutputStream(PipedInputStream)

Klasy podstawowe Unicode

- Reader
  - CharArrayReader(tablica bajtów)
  - StringReader(String)
  - FileReader(String lub File)
  - PipedReader(PipedOutputStream)
- Writer
  - CharArrayWriter(początkowa wielkość bufora)
  - StringWriter(String)
  - FileWriter(String lub File)
  - PipedWriter(PipedInputStream)

Klasy konwertujące:

InputStream —InputStreamReader→ Reader

OutputStream —OutputStreamWriter→ Writer

Klasy „dekoratorów”

- FilterInputStream(InputStream)
  - **DataInputStream – typy proste**
  - **BufferInputStream – buforowanie strumienia**
  - **LineNumberInputStream – z numerowaniem linii**
- FilterOutputStream(OutputStream)
  - **DataOutputStream**
  - **BufferedOutputStream**
  - **PrintStream**

Klasy „dekoratorów” Unicode

- FilterReader(Reader)
  - **BufferReader –buforowanie strumienia**
  - **LineNumberReader – z numerowaniem linii**
- FilterWriter(Writer)
  - **BufferedWriter**
  - **PrintWriter**

Przykłady

```
BufferedReader in = new BufferedReader(new FileReader("IOStreamDemo.java"));
String s, s2 = new String();
while((s = in.readLine())!= null)  s2 += s + "\n";
in.close();
```

```
try {
   BufferedReader in4 = new BufferedReader( new StringReader(s2));
   PrintWriter out1=new PrintWriter( new BufferedWriter( new
                              FileWriter("IODemo.out")));
   int lineCount = 1;
   while((s = in4.readLine()) != null ) out1.println(lineCount++ + ": " + s);
   out1.close();
  } catch(EOFException e) {
    System.err.println("End of stream");
}
```


Klasa RandomAccessFile

Metody: seek(), length(), read(), getFilePointer(), write(), close(),…

Korzystanie:

```
rf = new RandomAccessFile(”plik.dat”,”rw”);
rf.seek(40);
rf.writeDouble(23.34);
rf.close();
```

Standardowe strumienie we/wy

- System.in InputStream
- System.out PrintStream
- System.err PrintStream

- System.setIn(strin)
- System.setOut(strout)
- System.setErr(strerr)

Kompresja danych

- ZipOutputStream
- GZIPOutputStream
- CheckedOutputStream

- ZipInputStream
- GZIPInputStream
- CheckedInputStream

ZIP → JARs

Atomizacja wejścia

- StreamTokenizer (InputStream)
  - ordinaryChar(char), nextToken(), ttype, nval, sval, TT_EOF,TT_EOL,TT_NUMBER,TT_WORD
- StringTokenizer (String)
  - hasMoreTokens(), nextToken()

Przykład:

Program sprawdzający czy wszystkie nazwy klas i interfejsów są pisane od dużej litery. Wywołanie z parametrem: -a tworzy repozytorium klas.

ClassScanner (1)

```
import java.io.*;
import java.util.*;
class MultiStringMap extends HashMap {
 public void add(String key, String value) {
  if(!containsKey(key))  put(key, new ArrayList());
  ((ArrayList)get(key)).add(value);
 }
 public ArrayList getArrayList(String key) {
  if(!containsKey(key)) {
   System.err.println( "ERROR: can't find key: " + key);
   System.exit(1);
  }
  return (ArrayList)get(key);
 }
 public void printValues(PrintStream p) {
  Iterator k = keySet().iterator();
  while(k.hasNext()) {
   String oneKey = (String)k.next();
   ArrayList val = getArrayList(oneKey);
   for(int i = 0; i < val.size(); i++)   p.println((String)val.get(i));
  }
 }
}
```

ClassScanner (2)

```
public class ClassScanner {
  private File path;
  private String[] fileList;
  private Properties classes = new Properties();
  private MultiStringMap
    classMap = new MultiStringMap(),
    identMap = new MultiStringMap();
  private StreamTokenizer in;
  public ClassScanner() throws IOException {
    path = new File(".");
    fileList = path.list(new JavaFilter());
    for(int i = 0; i < fileList.length; i++) {
      System.out.println(fileList[i]);
      try {
        scanListing(fileList[i]);
      } catch(FileNotFoundException e) {
        System.err.println("Could not open " +
          fileList[i]);
      }
    }
  }
```


ClassScanner (3)

```
void scanListing(String fname)
 throws IOException {
  in = new StreamTokenizer( new BufferedReader( new FileReader(fname)));
  in.ordinaryChar('/');
  in.ordinaryChar('.');
  in.wordChars('_', '_');
  in.eolIsSignificant(true);
  while(in.nextToken() !=  StreamTokenizer.TT_EOF) {
    if(in.ttype == '/')  eatComments();
    else if(in.ttype ==  StreamTokenizer.TT_WORD) {
      if(in.sval.equals("class") ||  in.sval.equals("interface")) {
        // Get class name:
          while(in.nextToken() !=  StreamTokenizer.TT_EOF
                && in.ttype !=  StreamTokenizer.TT_WORD)  ;
          classes.put(in.sval, in.sval);
          classMap.add(fname, in.sval);
      }
      if(in.sval.equals("import") || in.sval.equals("package"))
        discardLine();
      else // It's an identifier or keyword
        identMap.add(fname, in.sval);
    }
  }
}
```

ClassScanner (4)

```
void discardLine() throws IOException {
   while(in.nextToken() !=  StreamTokenizer.TT_EOF && in.ttype !=
        StreamTokenizer.TT_EOL)
     ; // Throw away tokens to end of line
 }
 // StreamTokenizer's comment removal seemed
 // to be broken. This extracts them:
 void eatComments() throws IOException {
   if(in.nextToken() !=  StreamTokenizer.TT_EOF) {
     if(in.ttype == '/')  discardLine();
     else if(in.ttype != '*')  in.pushBack();
     else
       while(true) {
         if(in.nextToken() ==  StreamTokenizer.TT_EOF)
           break;
         if(in.ttype == '*')
           if(in.nextToken() !=  StreamTokenizer.TT_EOF && in.ttype == '/')
             break;
       }
   }
 }
```


ClassScanner (5)

```
public String[] classNames() {
    String[] result = new String[classes.size()];
    Iterator e = classes.keySet().iterator();
    int i = 0;
    while(e.hasNext()) result[i++] = (String)e.next();
    return result;
  }
  public void checkClassNames() {
    Iterator files = classMap.keySet().iterator();
    while(files.hasNext()) {
      String file = (String)files.next();
      ArrayList cls = classMap.getArrayList(file);
      for(int i = 0; i < cls.size(); i++) {
        String className = (String)cls.get(i);
        if(Character.isLowerCase(className.charAt(0)))
          System.out.println( "class capitalization error, file: "
            + file + ", class: " + className);
      }
    }
  }
```

ClassScanner (6)

```
public void checkIdentNames() {
   Iterator files = identMap.keySet().iterator();
   ArrayList reportSet = new ArrayList();
   while(files.hasNext()) {
     String file = (String)files.next();
     ArrayList ids = identMap.getArrayList(file);
     for(int i = 0; i < ids.size(); i++) {
       String id = (String)ids.get(i);
       if(!classes.contains(id)) {
         // Ignore identifiers of length 3 or
         // longer that are all uppercase
         // (probably static final values):
         if(id.length() >= 3 &&  id.equals( id.toUpperCase()))  continue;
         // Check to see if first char is upper:
         if(Character.isUpperCase(id.charAt(0))){
           if(reportSet.indexOf(file + id)  == -1){ // Not reported yet
             reportSet.add(file + id);
             System.out.println( "Ident capitalization error in:" + file + ", ident: " + id);
           }
         }
       }
     }
   }
}
```


ClassScanner (7)

```
static final String usage =
    "Usage: \n" +
    "ClassScanner classnames -a\n" +
    "\tAdds all the class names in this \n" +
    "\tdirectory to the repository file \n" +
    "\tcalled 'classnames'\n" +
    "ClassScanner classnames\n" +
    "\tChecks all the java files in this \n" +
    "\tdirectory for capitalization errors, \n" +
    "\tusing the repository file 'classnames'";
  private static void usage() {
    System.err.println(usage);
    System.exit(1);
  }
```

ClassScanner (8)

```
public static void main(String[] args)
 throws IOException {
  if(args.length < 1 || args.length > 2)
    usage();
  ClassScanner c = new ClassScanner();
  File old = new File(args[0]);
  if(old.exists()) {
    try {
      InputStream oldlist =
       new BufferedInputStream(
         new FileInputStream(old));
      c.classes.load(oldlist);
      oldlist.close();
    } catch(IOException e) {
      System.err.println("Could not open "
        + old + " for reading");
      System.exit(1);
    }
  }
  if(args.length == 1) {
    c.checkClassNames();
    c.checkIdentNames();
  }
  if(args.length == 2) {
      if(!args[1].equals("-a"))  usage();
      try {
        BufferedOutputStream out =
          new BufferedOutputStream(
            new FileOutputStream(args[0]));
        c.classes.store(out,
          "Classes found by ClassScanner.java");
        out.close();
      } catch(IOException e) {
        System.err.println("Could not write "+args[0]);
        System.exit(1);
      }
    }
  }
}
class JavaFilter implements FilenameFilter {
  public boolean accept(File dir, String name) {
    String f = new File(name).getName();
    return f.trim().endsWith(".java");
  }
}
```


Pytania

1. Zasady organizacji wejścia/wyjścia w języku Java.
2. Zastosowania polimorfizmu w strumieniach we/wy.

Koniec