Метод **String findInLine(Pattern pattern)** или **String findInLine(String pattern)** ищет заданный шаблон в следующей строке текста. Если шаблон найден, соответствующая ему подстрока извлекается из строки ввода. Если совпадений не найдено, то возвращается **null**.

Методы **String findWithinHorizon(Pattern pattern, int count)** и **String findWithinHorizon(String pattern, int count)** производят поиск заданного шаблона в ближайших **count** символах. Можно пропустить образец с помощью метода **skip(Pattern pattern)**.

Если в строке ввода найдена подстрока, соответствующая образцу **pattern**, метод **skip()** просто перемещается за нее в строке ввода и возвращает ссылку на вызывающий объект. Если подстрока не найдена, метод **skip()** генерирует исключение **NoSuchElementException**.

Архивация

Для хранения классов языка Java и связанных с ними ресурсов в языке Java используются сжатые архивные **jar**-файлы.

Для работы с архивами в спецификации Java существуют два пакета – **java.util.zip** и **java.util.jar** соответственно для архивов **zip** и **jar**. Различие форматов **jar** и **zip** заключается только в расширении архива zip. Пакет **java.util.jar** аналогичен пакету **java.util.zip**, за исключением реализации конструкторов и метода **void putNextEntry(ZipEntry e)** класса **JarOutputStream**. Ниже будет рассмотрен только пакет **java.util.jar**. Чтобы переделать все примеры на использование **zip**-архива, достаточно всюду в коде заменить **Jar** на **Zip**.

Пакет **java.util.jar** позволяет считывать, создавать и изменять файлы форматов **jar**, а также вычислять контрольные суммы входящих потоков данных.

Класс **JarEntry** (подкласс **ZipEntry**) используется для предоставления доступа к записям **jar**-файла. Наиболее важными методами класса являются:

void setMethod(int method) – устанавливает метод сжатия записи;

int getMethod() – возвращает метод сжатия записи;

void setComment(String comment) – устанавливает комментарий записи;

String getComment() – возвращает комментарий записи;

void setSize(long size) – устанавливает размер несжатой записи;

long getSize() – возвращает размер несжатой записи;

long getCompressedSize() – возвращает размер сжатой записи;

У класса **JarOutputStream** существует возможность записи данных в поток вывода в **jar**-формате. Он переопределяет метод **write()** таким образом, чтобы любые данные, записываемые в поток, предварительно сжимались. Основными методами данного класса являются:

void setLevel(int level) – устанавливает уровень сжатия. Чем больше уровень сжатия, тем медленней происходит работа с таким файлом;

`void putNextEntry(ZipEntry e)` – записывает в поток новую **`jar`**-запись. Этот метод переписывает данные из экземпляра **`JarEntry`** в поток вывода;

`void closeEntry()` – завершает запись в поток **`jar`**-записи и заносит дополнительную информацию о ней в поток вывода;

`void write(byte b[], int off, int len)` – записывает данные из буфера **`b`** начиная с позиции **`off`** длиной **`len`** в поток вывода;

`void finish()` – завершает запись данных **`jar`**-файла в поток вывода без закрытия потока;

`void close()` – закрывает поток записи.

```
/* пример # 12 : создание jar-архива: PackJar.java */
package chapt09;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
import java.util.jar.JarEntry;
import java.util.jar.JarOutputStream;
import java.util.zip.Deflater;

public class PackJar {
   public static void pack(String[] filesToJar,
                  String jarFileName, byte[] buffer) {
      try {
            JarOutputStream jos =
                  new JarOutputStream(
                        new FileOutputStream(jarFileName));
            // метод сжатия
            jos.setLevel(Deflater.DEFAULT_COMPRESSION);
            for (int i = 0; i < filesToJar.length; i++) {
                  System.out.println(i);
            jos.putNextEntry(new JarEntry(filesToJar[i]));

            FileInputStream in =
                        new FileInputStream(filesToJar[i]);
             int len;
              while ((len = in.read(buffer)) > 0)
                  jos.write(buffer, 0, len);
              jos.closeEntry();
              in.close();
            }
            jos.close();
      } catch (IllegalArgumentException e) {
            e.printStackTrace();
            System.err.println("Некорректный аргумент");
      } catch (FileNotFoundException e) {
            e.printStackTrace();
```

```
            System.err.println("Файл не найден");
        } catch (IOException e) {
            e.printStackTrace();
            System.err.println("Ошибка доступа");
        }
    }
    public static void main(String[] args) {
        System.out.println("Создание jar-архива");
        // массив файлов для сжатия
        String[] filesToJar = new String[2];
        filesToJar[0] = "chapt09//UseJar.java";
        filesToJar[1] = "chapt09//UseJar.class";
        byte[] buffer = new byte[1024];
        // имя полученного архива
        String jarFileName = "example.jar";
        pack(filesToJar, jarFileName, buffer);
    }
}
```

Класс **JarFile** обеспечивает гибкий доступ к записям, хранящимся в **jar**-файле. Это очень эффективный способ, поскольку доступ к данным осуществляется гораздо быстрее, чем при считывании каждой отдельной записи. Единственным недостатком является то, что доступ может осуществляться только для чтения. Метод **entries**() извлекает все записи из **jar**-файла. Этот метод возвращает список экземпляров **JarEntry** – по одной для каждой записи в **jar**- файле. Метод **getEntry(String name)** извлекает запись по имени. Метод **getInputStream()** создает поток ввода для записи. Этот метод возвращает поток ввода, который может использоваться приложением для чтения данных записи.

Класс **JarInputStream** читает данные в **jar**-формате из потока ввода. Он переопределяет метод **read()** таким образом, чтобы любые данные, считываемые из потока, предварительно распаковывались.

```
/* пример # 13 : чтение jar-архива: UnPackJar.java */
package chapt09;
import java.io.*;
import java.util.Enumeration;
import java.util.jar.JarEntry;
import java.util.jar.JarFile;

public class UnPackJar {
      private File destFile;
      // размер буфера для распаковки
      public final int BUFFER = 2048;

  public void unpack(String destinationDirectory,
                                          String nameJar) {
            File sourceJarFile = new File(nameJar);
            try {
            File unzipDestinationDirectory =
```

```
                    new File(destinationDirectory);
              // открытие zip-архива для чтения
        JarFile jFile = new JarFile(sourceJarFile);
        Enumeration jarFileEntries = jFile.entries();
        while (jarFileEntries.hasMoreElements()) {
              // извлечение текущей записи из архива
        JarEntry entry =
              (JarEntry) jarFileEntries.nextElement();

        String entryname = entry.getName();
              //entryname = entryname.substring(2);
        System.out.println("Extracting: " + entry);
              destFile =
        new File(unzipDestinationDirectory, entryname);
              // определение каталога
              File destinationParent =
                          destFile.getParentFile();
              // создание структуры каталогов
                    destinationParent.mkdirs();
              // распаковывание записи, если она не каталог
                    if (!entry.isDirectory()) {
                          writeFile(jFile, entry);
                    }
              }
              jFile.close();
        } catch (IOException ioe) {
              ioe.printStackTrace();
        }
    }
private void writeFile(JarFile jFile, JarEntry entry)
                          throws IOException {
        BufferedInputStream is =
              new BufferedInputStream(
                    jFile .getInputStream(entry));
        int currentByte;
        byte data[] = new byte[BUFFER];
        // запись файла на диск
        BufferedOutputStream dest =
              new BufferedOutputStream(
              new FileOutputStream(destFile), BUFFER);

    while ((currentByte = is.read(data, 0, BUFFER)) > 0){
              dest.write(data, 0, currentByte);
        }
        dest.flush();
        dest.close();
        is.close();
    }
```

```
    public static void main(String[] args) {
        System.out.println(
            "Извлечение данных из jar-архива");
        // расположение и имя архива
        String nameJar = "c:\\work\\example.jar";
        // куда файлы будут распакованы
        String destination = "c:\\temp\\";
        new UnPackJar().unpack(destination, nameJar);
    }
}
```

Задания к главе 9

Вариант A

В следующих заданиях требуется ввести последовательность строк из текстового потока и выполнить указанные действия. При этом могут рассматриваться два варианта:

- каждая строка состоит из одного слова;
- каждая строка состоит из нескольких слов.

Имена входного и выходного файлов, а также абсолютный путь к ним могут быть введены как параметры командной строки или храниться в файле.

1. В каждой строке найти и удалить заданную подстроку.
2. В каждой строке стихотворения Александра Блока найти и заменить заданную подстроку на подстроку иной длины.
3. В каждой строке найти слова, начинающиеся с гласной буквы.
4. Найти и вывести слова текста, для которых последняя буква одного слова совпадает с первой буквой следующего слова.
5. Найти в строке наибольшее число цифр, идущих подряд.
6. В каждой строке стихотворения Сергея Есенина подсчитать частоту повторяемости каждого слова из заданного списка и вывести эти слова в порядке возрастания частоты повторяемости.
7. В каждом слове сонета Вильяма Шекспира заменить первую букву слова на прописную.
8. Определить частоту повторяемости букв и слов в стихотворении Александра Пушкина.

Вариант B

Выполнить задания из варианта B главы 4, сохраняя объекты приложения в одном или нескольких файлах с применением механизма сериализации. Объекты могут содержать поля, помеченные как **`static`**, а также **`transient`**. Для изменения информации и извлечения информации в файле создать специальный класс-коннектор с необходимыми для выполнения этих задач методами.

Вариант C

При выполнении следующих заданий для вывода результатов создавать новую директорию и файл средствами класса **`File`**.

1. Создать и заполнить файл случайными целыми числами. Отсортировать содержимое файла по возрастанию.