ФАЙЛОВИЙ ВВІД-ВИВІД У МОВІ ПРОГРАМУВАННЯ JAVA

Питання 5.4.

Клас File

- Часто для роботи додатків необхідно взаємодіяти з файловою системою, що являє собою ієрархію файлів та папок, що розгалужується від кореневої (root) директорії.
  - Unix/Linux платформи (у т. ч. Android) об'єднують всі змонтовані (mounted) диски в єдину віртуальну файлову систему.
  - На противагу цьому, Windows асоціює окрему файлову систему для кожного активного диску.
- Доступ до відповідної файлової системи засобами Java здійснюється за допомогою класу java.io.File.
  - Існує можливість повернути всі кореневі директорії доступних файлових систем у вигляді масиву об'єктів File за допомогою методу listRoots().

```
import java.io.File;

public class DumpRoots
{
   public static void main(String[] args)
   {
      File[] roots = File.listRoots();
      for (File root: roots)
         System.out.println(root);
   }
}
```

```
C:\
D:\
E:\
```

- такий перелік буде залежати від підключення або від'єднання фізичних чи віртуальних дисків.
- Під Unix/Linux платформами на екрані відобразиться знак кореню віртуальної файлової системи (/).

Конструювання екземплярів File

- `File file1 = new File("/x/y");`
- `File file1 = new File("C:\\temp\\t.dat");`
- Перший рядок отримує доступ до файлу або директорії y, яка знаходиться в каталозі x.
  - Такий запис буде працювати і під Windows: кореневим каталогом буде поточна коренева директорія.
  - Перший символ «/» позначає кореневий каталог, а другий – сепаратор в ієрархії файлової системи.
- Другий рядок націлений на Windows-платформу, він задає кореневу директорію C: і вказує на файл t.dat у папці temp.
  - Символ-сепаратор для Windows – «\».
  - *Хорошим стилем програмування є завжди подвоювати символ-сепаратор з метою зменшення ризику появи помилок компілятора. У даному випадку таке подвоєння дозволяє уникнути трактування \t як символу табуляції рядка.*
- Кожна інструкція має абсолютний шлях;
  - додаткової інформації для знаходження файлу/папки не потрібно.
  - На відміну від цього, ***відносний шлях (relative pathname)*** стартує не з символу кореневої директорії, а від іншого шляху.
- **Зауважте!** Класи пакету java.io за замовчуванням працюють з відносними шляхами стосовно папки поточного користувача (робочої директорії), яка ідентифікується системною властивістю user.dir і зазвичай є папкою, в якій міститься віртуальна машина, що запущена.

Конструювання екземплярів File

- Екземпляри File містять абстрактні представлення файлу та шляху до папки, в якій він знаходиться (вони можуть і не існувати у файловій системі), зберігаючи абстрактні шляхи (*abstract pathnames*), які пропонують платформонезалежні представлення ієрархічних шляхів (views of hierarchical pathnames).
  - На противагу цьому, інтерфейси користувача та операційні системи використовують платформозалежні *pathname strings*, щоб іменувати файли та директорії.
- Абстрактний шлях складається з опційного платформозалежного рядку-префіксу, наприклад, специфікатор disk drive (/ для Unix/Linux або \\ для Windows Universal Naming Convention (UNC) pathname), а також послідовності рядкових назв.
  - Конвертація pathname string в/з абстрактного шляху платформозалежна.
  - Коли шлях в рядковому форматі конвертується в абстрактний шлях, назви в рядку можуть відокремлюватись за допомогою символу-сепаратору за замовчуванням або іншим символом-сепаратором, що підтримується базовою платформою (underlying platform).
  - Коли абстрактний шлях конвертується в pathname string, кожна назва відокремлюється від наступної за допомогою одної копії символу-сепаратора імен за замовчуванням (default name-separator character).

Конструювання екземплярів File

- *Символ-сепаратор імен за замовчуванням* визначається системною властивістю file.separator та доступний у публічних статичних полях separator і separatorChar класу File;
  - Перше поле зберігає символ в екземплярі java.lang.String, а друге – як char value.
- Клас File пропонує додаткові конструктори для інстанціювання.
  - Наприклад, наступні конструктори виконують злиття (merge) батьківського та дочірнього шляхів у combined pathnames, які зберігаються в об’єктах File:
    - File(String parent, String child) створює екземпляр File з батьківським та дочірнім шляхами у формі рядка.
    - File(File parent, String child) створює екземпляр File з батьківським та дочірнім шляхами типу File.
- В параметр parent конструкторів передається *parent pathname* – рядок, який складається з усіх компонентів шляху, крім останньої назви, яка задається child.
  - File file3 = new File("prj/books/", "ljfad3");
  - Конструктор зливає батьківський шлях prj/books/ з дочірнім шляхом ljfad3 в єдиний шлях prj/books/ljfad3.

Збережені абстрактні шляхи (Stored Abstract Pathnames)

- Оскільки File(String pathname), File(String parent, String child) та File(File parent, String child) не визначають некоректності аргументів, крім викидання NullPointerException, коли pathname або child дорівнюють null, необхідно ретельно задавати шляхи.
  - Прагніть задавати шляхи, які будуть коректними для всіх платформ, на яких буде запускатись додаток.
  - Наприклад, замість жорстко закодованого drive specifier (як C:) у шляху, використовуйте кореневі каталоги від методу listRoots().
  - Ще краще тримати шляхи відносними щодо current user/working directory (отримується від системної властивості user.dir).
- Після створення об’єкту типу File його можна опитати, щоб дізнатись про збережений в ньому абстрактний шлях, викликаючи методи з наступних слайдів

Методи File, щоб дізнатись Stored Abstract Pathname

Метод	Опис
File getAbsoluteFile()	Повертає абсолютну форму абстрактного шляху File-об’єкта. Еквівалентний до new File(this.getAbsolutePath())
String getAbsolutePath()	Повертає абсолютного рядкове представлення абстрактного шляху File-об’єкта. Якщо воно вже абсолютне, рядок повертається так, ніби було викликано getPath(). Якщо абстрактний шлях порожній, повертається рядкове представлення шляху до поточної директорії користувача (за допомогою user.dir). Інакше абстрактний шлях утворюється платформозалежним способом. На Unix/Linux відносний шлях перетворюється в абсолютний, поєднуючись з поточною директорією користувача. На Windows – з поточною директорією на логічному диску або користувацькою директорією, якщо диску немає.
File getCanonicalFile()	Повертає канонічну (найпростішу, абсолютну та унікальну) форму абстрактного шляху File-об’єкту. Метод викидає java.io.IOException, коли трапляється помилка вводу-виводу (створення канонічного шляху може вимагати запитів до файлової системи); метод відповідає коду: new File(this.getCanonicalPath())

Method	Description
`String getCanonicalPath()`	Returns the canonical pathname string of this `File` object's abstract pathname. This method first converts this pathname to absolute form when necessary, as if by invoking `getAbsolutePath()`, and then maps it to its unique form in a platform-dependent way. Doing so typically involves removing redundant names such as . and .. from the pathname, resolving symbolic links (on Unix/Linux platforms), and converting drive letters to a standard case (on Windows platforms). This method throws `IOException` when an I/O error occurs (creating the canonical pathname may require filesystem queries).
`String getName()`	Returns the filename or directory name denoted by this `File` object's abstract pathname. This name is the last in a pathname's name sequence. The empty string is returned when the pathname's name sequence is empty.
`String getParent()`	Returns the parent pathname string of this `File` object's pathname, or returns null when this pathname doesn't name a parent directory.
`File getParentFile()`	Returns a `File` object storing this `File` object's abstract pathname's parent abstract pathname; returns null when the parent pathname isn't a directory.
`String getPath()`	Converts this `File` object's abstract pathname into a pathname string where the names in the sequence are separated by the character stored in `File`'s `separator` field. Returns the resulting pathname string.
`boolean isAbsolute()`	Returns true when this `File` object's abstract pathname is absolute; otherwise, returns false when it's relative. The definition of absolute pathname is system dependent. On Unix/Linux platforms, a pathname is absolute when its prefix is `/`. On Windows platforms, a pathname is absolute when its prefix is a drive specifier followed by `\` or when its prefix is `\\`.
`String toString()`	A synonym for `getPath()`.

Отримання інформації про файл/директорію

```
import java.io.File;
import java.io.IOException;

import java.util.Date;

public class FileDirectoryInfo
{
   public static void main(final String[] args) throws IOException
   {
      if (args.length != 1)
      {
         System.err.println("usage: java FileDirectoryInfo pathname");
         return;
      }
      File file = new File(args[0]);
      System.out.println("About " + file + ":");
      System.out.println("Exists = " + file.exists());
      System.out.println("Is directory = " + file.isDirectory());
      System.out.println("Is file = " + file.isFile());
      System.out.println("Is hidden = " + file.isHidden());
      System.out.println("Last modified = " + new Date(file.lastModified()));
      System.out.println("Length = " + file.length());
   }
}
```

- Нехай матимемо three-byte файл x.dat.
- Коли записати java FileDirectoryInfo x.dat, спостерігається наступний вивід:

```
About x.dat:
Exists = true
Is directory = false
Is file = true
Is hidden = false
Last modified = Mon Oct 14 15:31:04 CDT 2013
Length = 3
```

Отримування інформації про вільне місце на диску

- ***Розділ (partition)*** – частина пам'яті для файлової системи, специфічна для конкретної платформи, наприклад, C:\.
  - Отримування об'єму вільного дискового простору важливе для інсталяторів та інших додатків.
  - До Java 6 єдиним портативним способом виконати це завдання було вгадування шляхом створення файлів різних розмірів.
  - Java 6 додала в клас File методи long getFreeSpace(), long getTotalSpace() і long getUsableSpace(), які повертають інформацію про простір розділу, описаного абстрактним шляхом екземпляру File.
- Android також підтримує ці методи:
  - ***long getFreeSpace()*** повертає кількість нерозподілених (unallocated) байтів розділу, визначеного абстрактним шляхом екземпляру File; якщо абстрактний шлях не містить назви розділу, повертає 0.
  - ***long getTotalSpace()*** повертає розмір розділу (в байтах), що визначається з abstract pathname цього об'єкту File; повертає 0, коли abstract pathname не містить назви розділу.
  - ***long getUsableSpace()*** повертає кількість байтів, доступних віртуальній машині з розділу, визначеного абстрактним шляхом об'єкту File; повертає 0, коли абстрактний шлях не містить назви розділу.
  - Хоч getFreeSpace() та getUsableSpace() здаються еквівалентними, проте відрізняються так: на відміну від getFreeSpace(), getUsableSpace() перевіряє дозволи на запис та інші обмеження платформи, повертаючи більш точну оцінку.

Методи getFreeSpace() та getUsableSpace()

- Повертають hint (not a guarantee), який Java-додаток може використовувати всі (або більшість) неалокованих або доступних байтів.
  - Ці значення будуть підказкою, оскільки стороння програма, що працює поза ВМ, може виділити для себе певний об’єм пам’яті, що змінить відповідно і кількість незайнятої та доступної пам’яті, що повертається цими методами.
- Демонстрація методів на наступному слайді.
  - Після отримання масиву всіх доступних roots файлової системи, додаток отримує та виводить вільне, повне та доступне для використання (usable) місце для кожного розділу, визначеного в масиві.

Виведення вільного, доступного та повного простору для всіх розділів

```
import java.io.File;

public class PartitionSpace
{
   public static void main(String[] args)
   {
      File[] roots = File.listRoots();
      for (File root: roots)
      {
         System.out.println("Partition: " + root);
         System.out.println("Free space on this partition = " +
                            root.getFreeSpace());
         System.out.println("Usable space on this partition = " +
                            root.getUsableSpace());
         System.out.println("Total space on this partition = " +
                            root.getTotalSpace());
         System.out.println("***");
      }
   }
}
```

- Варіант виводу

```
Partition: C:\
Free space on this partition = 374407311360
Usable space on this partition = 374407311360
Total space on this partition = 499808989184
***
Partition: D:\
Free space on this partition = 0
Usable space on this partition = 0
Total space on this partition = 0
***
Partition: E:\
Free space on this partition = 2856464384
Usable space on this partition = 285646438
Total space on this partition = 8106606592
***
Partition: F:\
Free space on this partition = 0
Usable space on this partition = 0
Total space on this partition = 0
***
```

Методи File для отримування вмісту папок

Method	Description
`String[] list()`	Returns a potentially empty array of strings naming the files and directories in the directory denoted by this `File` object's abstract pathname. If the pathname doesn't denote a directory, or if an I/O error occurs, this method returns null. Otherwise, it returns an array of strings, one string for each file or directory in the directory. Names denoting the directory itself and the directory's parent directory are not included in the result. Each string is a filename rather than a complete path. Also, there is no guarantee that the name strings in the resulting array will appear in alphabetical or any other order.
`String[] list(FilenameFilter filter)`	A convenience method for calling `list()` and returning only those `Strings` that satisfy `filter`.
`File[] listFiles()`	A convenience method for calling `list()`, converting its array of `Strings` to an array of `Files`, and returning the `Files` array.
`File[] listFiles(FileFilter filter)`	A convenience method for calling `list()`, converting its array of `Strings` to an array of `Files`, but only for those `Strings` that satisfy `filter`, and returning the `Files` array.
`File[] listFiles(FilenameFilter filter)`	A convenience method for calling `list()`, converting its array of `Strings` to an array of `Files`, but only for those `Strings` that satisfy `filter`, and returning the `Files` array.

- Перевантажені методи list() повертають масиви Strings, що позначають назви файлу та директорії.
  - Другий метод дозволяє повертати лише names of interest (наприклад, лише ті файли, назви яких закінчуються розширенням .txt) за допомогою java.io.FilenameFilter-based filter object.
- Інтерфейс FilenameFilter оголошує один метод boolean accept(File dir, String name), який викликається для кожного файлу/каталогу, що розміщується в директорії, визначеній шляхом з об’єкту типу File:
  - dir визначає батьківську частину шляху (шлях до папки).
  - name вказує назву кінцевої папки чи назву файлу.
- Метод accept() використовує аргументи, передані цими параметрами, щоб визначити, чи задовольняє файл або каталог критерій допустимості.
  - Повертає true, коли назва файлу/каталогу має включатись в returned array; інакше метод поверне false.

Перелік конкретних назв

```
import java.io.File;
import java.io.FilenameFilter;

public class Dir
{
   public static void main(final String[] args)
   {
      if (args.length != 2)
      {
         System.err.println("usage: java Dir dirpath ext");
         return;
      }
      File file = new File(args[0]);
      FilenameFilter fnf = new FilenameFilter()
                           {
                              @Override
                              public boolean accept(File dir, String name)
                              {
                                 return name.endsWith(args[1]);
                              }
                           };
      String[] names = file.list(fnf);
      for (String name: names)
         System.out.println(name);
   }
}
```

- Варіант виводу

```
bfsvc.exe
explorer.exe
fveupdate.exe
HelpPane.exe
hh.exe
notepad.exe
regedit.exe
splwow64.exe
twunk_16.exe
twunk_32.exe
winhlp32.exe
write.exe
```

- Перевантажені методи listFiles() повертають масив File.
  - У значній мірі методи аналогічні відповідним list().
  - Проте listFiles(FileFilter) працює по-іншому.
- Інтерфейс java.io.FileFilter оголошує один метод boolean accept(String pathname), який викликається для кожного файлу/каталогу, що розміщено в директорії, визначеній шляхом до об’єкту File;
  - Аргумент, переданий to pathname визначає повний шлях до файлу або директорії.
- Метод accept() використовує цей аргумент, щоб визначити, чи задовольняє файл або директорія критерій допустимості
  - Повертає true, коли назва файлу/папки має включатись у returned array; інакше повертає false.
- **Зауважте!** Оскільки кожен метод accept() інтерфейсу виконує ту ж задачу, який інтерфейс використовувати?
  - Якщо віддаєте перевагу шляху, розбитому на компоненти directory та name, використовуйте FilenameFilter.
  - Інакше, при complete pathname, беріть FileFilter; можна завжди викликати getParent() та getName() для отримання цих компонентів.

Створення та маніпулювання файлами та папками

Method	Description
`boolean createNewFile()`	Atomically creates a new, empty file named by this `File` object's abstract pathname if and only if a file with this name doesn't yet exist. The check for file existence (and the creation of the file when it doesn't exist) is a single operation that's atomic with respect to all other filesystem activities that might affect the file. This method returns true when the named file doesn't exist and was successfully created, and returns false when the named file already exists. It throws `IOException` when an I/O error occurs.
`static File createTempFile(String prefix, String suffix)`	Creates an empty file in the default temporary file directory using the given `prefix` and `suffix` to generate its name. This overloaded class method calls its three-parameter variant, passing `prefix`, `suffix`, and `null` to this other method, and returning the other method's return value.
`static File createTempFile(String prefix, String suffix, File directory)`	Creates an empty file in the specified `directory` using the given `prefix` and `suffix` to generate its name. The name begins with the character sequence specified by `prefix` and ends with the character sequence specified by `suffix`; `.tmp` is used as the suffix when `suffix` is `null`. This method returns the created file's pathname when successful. It throws `java.lang.IllegalArgumentException` when `prefix` contains fewer than three characters and `IOException` when the file couldn't be created.
`boolean delete()`	Deletes the file or directory denoted by this `File` object's pathname. Returns true when successful; otherwise, returns false. If the pathname denotes a directory, the directory must be empty in order to be deleted.

`void deleteOnExit()`	Requests that the file or directory denoted by this `File` object's abstract pathname be deleted when the virtual machine terminates. Reinvoking this method on the same `File` object has no effect. Once deletion has been requested, it's not possible to cancel the request. Therefore, this method should be used with care.
`boolean mkdir()`	Creates the directory named by this `File` object's abstract pathname. Returns true when successful; otherwise, returns false.
`boolean mkdirs()`	Creates the directory and any necessary intermediate directories named by this `File` object's abstract pathname. Returns true when successful; otherwise, returns false.
`boolean renameTo(File dest)`	Renames the file denoted by this `File` object's abstract pathname to `dest`. Returns true when successful; otherwise, returns false. This method throws `NullPointerException` when `dest` is `null`. Many aspects of this method's behavior are platform dependent. For example, the rename operation might not be able to move a file from one filesystem to another, the operation might not be atomic, or it might not succeed when a file with the destination pathname already exists. The return value should always be checked to make sure that the rename operation was successful.
`boolean setLastModified(long time)`	Sets the last-modified time of the file or directory named by this `File` object's abstract pathname. Returns true when successful; otherwise, returns false. This method throws `IllegalArgumentException` when `time` is negative. All platforms support file-modification times to the nearest second, but some provide more precision. The `time` value will be truncated to fit the supported precision. If the operation succeeds and no intervening operations on the file take place, the next call to `lastModified()` will return the (possibly truncated) `time` value passed to this method.

- Припустимо, розробляється текстовий редактор, в якому користувач буде відкривати текстовий файл та вносити в нього зміни.
  - Поки користувач явно не збереже зміни у файлі, бажано залишати файл незмінним.
  - Оскільки користувач не хоче втрачати зміни при відмові додатку або перебоїв роботи комп’ютера, додаток проектується з урахуванням збереження змін у тимчасовому файлі кожні кілька хвилин.
  - Можна використовувати перевантажені методи createTempFile() для створення тимчасового файлу.
  - Якщо не задавати папку, в якій зберігатиметься файл, він буде створюватись у директорії, визначеній системною властивістю java.io.tmpdir system property.
  - Після того, як користувач явно збереже зміни або відмінить їх, бажано видалити тимчасовий файл.
  - Метод deleteOnExit() дозволяє записати тимчасовий файл на видалення; виконує видалення віртуальна машина без crash/power втрат.

Експериментуємо з тимчасовими файлами

```
import java.io.File;
import java.io.IOException;

public class TempFileDemo
{
   public static void main(String[] args) throws IOException
   {
      System.out.println(System.getProperty("java.io.tmpdir"));
      File temp = File.createTempFile("text", ".txt");
      System.out.println(temp);
      temp.deleteOnExit();
   }
}
```

- Після виводу місця розташування тимчасових файлів TempFileDemo створює тимчасовий файл, назва якого починається з text і закінчується розширенням .txt.
- Потім TempFileDemo виводить назву тимчасового файлу та реєструє його на видалення при успішному закінченні роботи додатку.

```
C:\Users\Owner\AppData\Local\Temp\
C:\Users\Owner\AppData\Local\Temp\text3173127870811188221.txt
```

Встановлення та отримання дозволів

- У Java 1.2 з’явився метод boolean setReadOnly() класу File для позначення файлу чи папки як read-only.
  - Проте метод для зворотних дій додано не було.
  - До виходу Java 6 клас File не пропонував способу управляти дозволами на read, write, execute абстрактного шляху.
- У Java 6 (клас File) було додано методи
  - boolean setExecutable(boolean executable),
  - boolean setExecutable(boolean executable, boolean ownerOnly),
  - boolean setReadable(boolean readable),
  - boolean setReadable(boolean readable, boolean ownerOnly),
  - boolean setWritable(boolean writable),
  - boolean setWritable(boolean writable, boolean ownerOnly)

для підтримки дозволів для власника чи будь-кого на виконання, зчитування та запис файлу, визначеного абстрактним шляхом.

Android також підтримує ці методи

- **boolean setExecutable(boolean executable, boolean ownerOnly)**
- **boolean setReadable(boolean readable, boolean ownerOnly)**
- **boolean setWritable(boolean writable, boolean ownerOnly)** вмикає(перший параметр - true) чи вимикає (false) дозвіл на запуск/ зчитування/ запис тільки для власника (ownerOnly=true) чи будь-кого(false).
  - Коли файлова система не відрізняє власника та всіх решту, цей дозвіл завжди застосовується для всіх.
  - Повертає true, коли операція вдала.
  - Повертає false, коли користувач не має дозволу змінювати дозволи на доступ до abstract pathname або коли перший параметр = false і файлова система не реалізує дозвіл на виконання/зчитування/запис.

- **boolean setExecutable(boolean executable)**
- **boolean setReadable(boolean readable)**
- **boolean setWritable(boolean writable)** викликає попередні методи для задавання дозволу на виконання/зчитування/запису для власника.
  - вмикає (readable= true) чи вимикає (false) дозвіл для власника (true для ownerOnly) чи будь-кого (false).
  - Коли файлова система не відрізняє власника від решти користувачів, цей дозвіл завжди надається всім.
  - Повертає true, коли операція вдала.
  - Повертає false, коли користувач не має дозволу на зміну дозволів на доступ чи коли параметр=false і відсутності дозволу на виконання/зчитування/запис у файловій системі.

```
import java.io.File;

public class Permissions
{
   public static void main(String[] args)
   {
      if (args.length != 1)
      {
         System.err.println("usage: java Permissions filespec");
         return;
      }
      File file = new File(args[0]);
      System.out.println("Checking permissions for " + args[0]);
      System.out.println("  Execute = " + file.canExecute());
      System.out.println("  Read = " + file.canRead());
      System.out.println("  Write = " + file.canWrite());
   }
}
```

- У Java 6 також переглянуто методи `boolean canRead()` та `boolean canWrite()` і додано `boolean canExecute()`, що повертає дозволи доступу для абстрактного шляху.
  - Методи повертають true, коли об’єкт файлу чи папки, що визначається абстрактним шляхом, існує, а підходящий дозвіл надано.
  - Наприклад, canWrite() повертає true, коли абстрактний шлях існує, а додаток має дозвіл на запис файлу.
  - Методи canRead(), canWrite(), canExecute() можна використовувати при реалізації утиліти для ідентифікації дозволів для довільного файлу чи папки.

```
Checking permissions for x
  Execute = true
  Read = true
  Write = false
```

Огляд додаткових можливостей

Method	Description
`int compareTo(File pathname)`	Compares two pathnames lexicographically. The ordering defined by this method depends upon the underlying platform. On Unix/Linux platforms, alphabetic case is significant when comparing pathnames; on Windows platforms, alphabetic case is insignificant. Returns zero when `pathname`'s abstract pathname equals this `File` object's abstract pathname, a negative value when this `File` object's abstract pathname is less than `pathname`, and a positive value otherwise. To accurately compare two `File` objects, call `getCanonicalFile()` on each `File` object and then compare the returned `File` objects.
`boolean equals(Object obj))`	Compares this `File` object with `obj` for equality. Abstract pathname equality depends upon the underlying platform. On Unix/Linux platforms, alphabetic case is significant when comparing pathnames; on Windows platforms, alphabetic case is insignificant. Returns true if and only if `obj` is not `null` and is a `File` object whose abstract pathname denotes the same file/directory as this `File` object's abstract pathname.
`int hashCode()`	Calculates and returns a hash code for this pathname. This calculation depends upon the underlying platform. On Unix/Linux platforms, a pathname's hash code equals the exclusive OR of its pathname string's hash code and decimal value 1234321. On Windows platforms, the hash code is the exclusive OR of the lowercased pathname string's hash code and decimal value 1234321. The current *locale* (geographical, political, or cultural region) is not taken into account when lowercasing the pathname string.

Порівняння файлів

```
import java.io.File;
import java.io.IOException;

public class Compare
{
   public static void main(String[] args) throws IOException
   {
      if (args.length != 2)
      {
         System.err.println("usage: java Compare filespec1 filespec2");
         return;
      }

      File file1 = new File(args[0]);
      File file2 = new File(args[1]);
      System.out.println(file1.compareTo(file2));
      System.out.println(file1.getCanonicalFile()
                              .compareTo(file2.getCanonicalFile()));
   }
}
```

- File реалізує метод compareTo() інтерфейсу java.lang.Comparable та перевизначає методи equals() та hashCode().

```
53
0
```

Клас RandomAccessFile

- Файли можуть створюватись і/або відкриватись за допомогою довільного доступу, за якого може відбуватись суміш зчитувань і записів, поки файл не закрито.
  - Java підтримує довільний доступ за допомогою конкретного класу java.io.RandomAccessFile, який використовується і в розробці під Android.
  - Наприклад, для зчитування RAW ресурсного файлу додатку
- Конструктори RandomAccessFile
  - **RandomAccessFile(String pathname, String mode)**
  - **RandomAccessFile(File file, String mode)** створює та відкриває новий файл, якщо він не існує, або відкриває існуючий файл.
    - Файл визначається через шлях (1) або абстрактний шлях (2) до нього та створюється/відкривається згідно з режимом доступу.
    - Режим конструктора – один зі списку: "r", "rw", "rws" чи " rwd"; інакше викидається виключення IllegalArgumentException.
    - "r" інформує конструктор про відкриття файлу тільки для зчитування. Будь-яка спроба записати дані викличе екземпляр класу IOException.
    - "rw" говорить конструктору створити та відкрити новий файл, якщо його не існує, для зчитування та запису або для відкриття існуючого файлу з тим же режимом.
    - "rwd" аналогічний попередньому, проте кожне оновлення вмісту файлу повинно записуватись синхронно на underlying storage device.
    - "rws" аналогічний попередньому, проте зачіпає і метадані щодо синхронного запису.

Зауваження

- Метаданими файлу називають дані про файл, а не його вміст.
  - До метаданих відносять розмір (length) файлу, час останнього редагування тощо.
  - Режими "rwd" та "rws" забезпечують стан, в якому будь-яка операція запису в файл на локальному носієві відбувається, що гарантує відсутність втрат критичних даних, коли операційна система збоїть.
  - Для файлів не з локального сховища даних гарантії немає.
- Операції довільного доступу з файлом у режимі "rwd" чи "rws" повільніші, ніж у режимі "rw".
  - Конструктори викидають FileNotFoundException, коли режим – "r", а визначений за допомогою шляху файл не може бути відкритим (не існує або це папка) чи в режимі "rw" за умови, що шлях read-only або directory.

Демонстрація другого конструктора при спробі відкрити існуючий файл

- Файл відкривається для зчитування в режимі "r":
  - RandomAccessFile raf = new RandomAccessFile("employee.dat", "r");
  - Файл з довільним доступом пов’язаний з вказівником на файл (*file pointer*) – курсором, що ідентифікує місце знаходження наступного байту для запису чи зчитування.
  - Коли відкривається існуючий файл, file pointer встановлюється на перший байт із зсувом (offset) 0.
  - Вказівник на файл також прирівнюється нулю, коли файл створюється.
- Операції зчитування та запису починаються з місця, на яке показує file pointer, та поширюються на задану кількість байтів.
  - Операції, що записують поточний файл до його кінця, спричиняють розширення файлу.
  - Ці операції продовжуються, поки файл не закрито.

Методи класу RandomAccessFile

Method	Description
`void close()`	Closes the file and releases any associated platform resources. Subsequent writes or reads result in `IOException`. Also, the file cannot be reopened with this `RandomAccessFile` object. This method throws `IOException` when an I/O error occurs.
`FileDescriptor getFD()`	Returns the file's associated file descriptor object. This method throws `IOException` when an I/O error occurs.
`long getFilePointer()`	Returns the file pointer's current zero-based byte offset into the file. This method throws `IOException` when an I/O error occurs.
`long length()`	Returns the length (measured in bytes) of the file. This method throws `IOException` when an I/O error occurs.
`int read()`	Reads and returns (as an `int` in the range 0 to 255) the next byte from the file or returns -1 when the end of the file is reached. This method blocks when no input is available and throws `IOException` when an I/O error occurs.
`int read(byte[] b)`	Reads up to `b.length` bytes of data from the file into byte array `b`. This method blocks until at least 1 byte of input is available. It returns the number of bytes read into the array, or returns -1 when the end of the file is reached. It throws `NullPointerException` when `b` is `null` and `IOException` when an I/O error occurs.

Методи класу RandomAccessFile

`char readChar()`	Reads and returns a character from the file. This method reads 2 bytes from the file starting at the current file pointer. If the bytes read, in order, are `b1` and `b2`, where 0 <= `b1`, `b2` <= 255, the result is equal to `(char) ((b1 << 8) | b2)`. This method blocks until the 2 bytes are read, the end of the file is detected, or an exception is thrown. It throws `java.io.EOFException` (a subclass of `IOException`) when the end of the file is reached before reading both bytes, and `IOException` when an I/O error occurs.
`int readInt()`	Reads and returns a 32-bit integer from the file. This method reads 4 bytes from the file starting at the current file pointer. If the bytes read, in order, are `b1`, `b2`, `b3`, and `b4`, where 0 <= `b1`, `b2`, `b3`, `b4` <= 255, the result is equal to `(b1 << 24) | (b2 << 16) | (b3 << 8) | b4`. This method blocks until the 4 bytes are read, the end of the file is detected, or an exception is thrown. It throws `EOFException` when the end of the file is reached before reading the 4 bytes and `IOException` when an I/O error occurs.
`void seek(long pos)`	Sets the file pointer's current offset to `pos` (which is measured in bytes from the beginning of the file). If the offset is set beyond the end of the file, the file's length doesn't change. The file length will only change by writing after the offset has been set beyond the end of the file. This method throws `IOException` when the value in `pos` is negative or when an I/O error occurs.

Method	Description
`void setLength(long newLength)`	Sets the file's length. If the present length as returned by `length()` is greater than `newLength`, the file is truncated. In this case, if the file offset as returned by `getFilePointer()` is greater than `newLength`, the offset will be equal to `newLength` after `setLength()` returns. If the present length is smaller than `newLength`, the file is extended. In this case, the contents of the extended portion of the file are not defined. This method throws `IOException` when an I/O error occurs.
`int skipBytes(int n)`	Attempts to skip over `n` bytes. This method skips over a smaller number of bytes (possibly zero) when the end of file is reached before `n` bytes have been skipped. It doesn't throw `EOFException` in this situation. If `n` is negative, no bytes are skipped. The actual number of bytes skipped is returned. This method throws `IOException` when an I/O error occurs.
`void write(byte[] b)`	Writes `b.length` bytes from byte array `b` to the file starting at the current file pointer position. This method throws `IOException` when an I/O error occurs.
`void write(int b)`	Writes the lower 8 bits of `b` to the file at the current file pointer position. This method throws `IOException` when an I/O error occurs.
`void writeChars(String s)`	Writes string `s` to the file as a sequence of characters starting at the current file pointer position. This method throws `IOException` when an I/O error occurs.
`void writeInt(int i)`	Writes 32-bit integer `i` to the file starting at the current file pointer position. The 4 bytes are written with the high byte first. This method throws `IOException` when an I/O error occurs.

Зауваження

- Назва більшості методів пояснює їх роботу.
  - Проте докладніше розглянемо метод getFD().
- Методи з префіксом read класу RandomAccessFile та метод skipBytes() походять з інтерфейсу java.io.DataInput, що реалізується класом.
  - Методи з префіксом write класу RandomAccessFile походять від інтерфейсу java.io.DataOutput, що реалізується даним класом.
  - Після відкриття файлу створює платформозалежну структуру для представлення файлу.
- Дескриптор (handle) цієї структури зберігається в екземплярі java.io.FileDescriptor, що повертає getFD().
  - Дескриптор – це ідентифікатор, який Java передає базовій платформі, щоб визначити, в даному випадку, конкретний відкритий файл, для якого існує потреба у виконанні базовою платформою файлової операції.

Клас FileDescriptor

- Невеликий клас, що оголошує три константи: in, out, err.
  - Вони дають можливість System.in, System.out та System.err забезпечувати доступ до стандартних потоків вводу, виводу та error streams.
- Пари методів з класу FileDescriptor:
  - **void sync()** вказує базовій платформі очистити (*flush)* вміст буферів виводу відкритого файлу із записом інформації на відповідний локальний диск.
    - Вихід з sync() відбувається після запису всіх змінених даних та атрибутів на відповідний пристрій.
    - Викидає java.io.SyncFailedException, коли буфери неможливо очистити або платформа не може гарантувати синхронізацію всіх буферів з фізичними пристроями.
  - **boolean valid()** визначає валідність об’єкту-дескриптору файлу.
    - Повертає true, коли об’єкт-дескриптор файлу представляє відкритий файл або інше активне I/O connection;
    - Інакше повертає false.
- Записані у відкритий файл дані наприкінці зберігаються в буфери виводу базової платформи.
  - Коли буфери вщерть заповнюються, платформа empties them на диск.
  - Буфери покращують продуктивність, оскільки доступ до диску повільний.
  - Проте при запису даних у файл з довільним доступом, який було відкрито в режимі "rwd" чи "rws", дані з кожної операції запису записуються напряму на диск.
  - Таким чином, операції запису повільніші, ніж в режимі "rw".

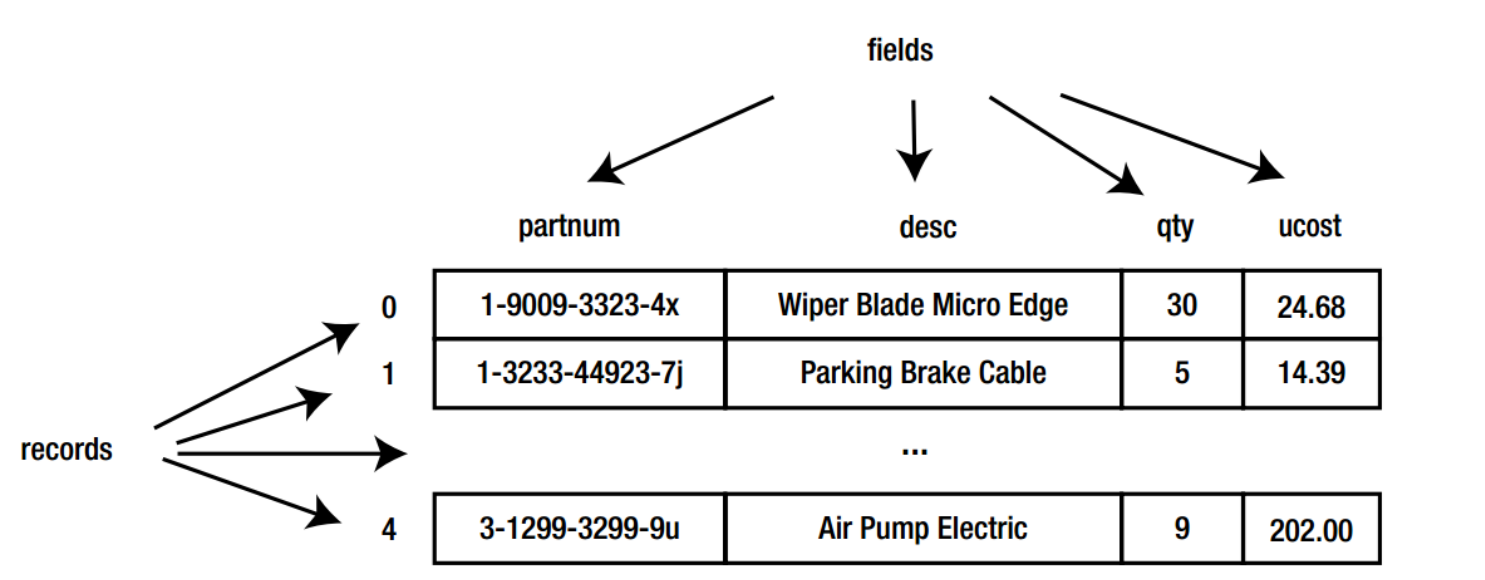
Ситуація з комбінацією запису даних через буфери виводу та напряму на диск

- Сценарій відкриття файлу в режимі "rw" та вибірковому виклику методу sync() класу FileDescriptor.

```
RandomAccessFile raf = new RandomAccessFile("employee.dat", "rw");
FileDescriptor fd = raf.getFD();
// Perform a critical write operation.
raf.write(...);
// Synchronize with underlying disk by flushing platform's output buffers to disk.
fd.sync();
// Perform non-critical write operation where synchronization isn't necessary.
raf.write(...);
// Do other work.
// Close file, emptying output buffers to disk.
raf.close();
```

БД на плоских файлах (*flat file database*)

- Клас RandomAccessFile корисний при створенні БД на плоских файлах – файлу, вміст якого організовано в записи та поля.
- **Зауважте!** Термін поле (*field)* також відноситься до змінних у класі.
  - У даному контексті можна розглядати field-змінну як аналог до атрибуту поля в записі (рядку таблиці).
  - БД на плоских файлах зазвичай організує свій вміст у послідовність записів (records) фіксованої довжини.
  - Кожен запис далі організується в одне або кілька полів фіксованого розміру.



```
import java.io.IOException;
import java.io.RandomAccessFile;

public class PartsDB
{
   public final static int PNUMLEN = 20;
   public final static int DESCLEN = 30;
   public final static int QUANLEN = 4;
   public final static int COSTLEN = 4;

   private final static int RECLEN = 2 * PNUMLEN + 2 * DESCLEN + QUANLEN + COSTLEN;
   private RandomAccessFile raf;

   public PartsDB(String pathname) throws IOException
   {
      raf = new RandomAccessFile(pathname, "rw");
   }

   public void append(String partnum, String partdesc, int qty, int ucost)
      throws IOException
   {
      raf.seek(raf.length());
      write(partnum, partdesc, qty, ucost);
   }
```

```
   public void close()
   {
      try
      {
         raf.close();
      }
      catch (IOException ioe)
      {
         System.err.println(ioe);
      }
   }

   public int numRecs() throws IOException
   {
      return (int) raf.length() / RECLEN;
   }
```

Реалізація частин БД на плоских файлах

- *Клас PartsDB спочатку оголошує константи, які ідентифікують довжини рядків та цілочисельні поля.*
  - *Потім – константа, що обчислює розмір запису в байтах (символи по 2 байти).*
- Поле raf – екземпляр класу RandomAccessFile, конструктор якого створює/відкриває новий або існуючий файл («rw»).
  - Далі PartsDB оголошує методи append(), close(), numRecs(), select(), update().
  - Метод append() спочатку викликає length() та seek().
  - Так забезпечуємо позиціонування файлового вказівника на кінець файлу перед викликом приватного методу write() для записування record containing this method's arguments.
  - Метод close() з RandomAccessFile може викидати IOException.
  - Оскільки це трапляється рідко, вирішено обробляти виняток у методі close() класу PartDB.

```
public Part select(int recno) throws IOException
{
   if (recno < 0 || recno >= numRecs())
      throw new IllegalArgumentException(recno + " out of range");
   raf.seek(recno * RECLEN);
   return read();
}

public void update(int recno, String partnum, String partdesc, int qty,
                   int ucost) throws IOException
{
   if (recno < 0 || recno >= numRecs())
      throw new IllegalArgumentException(recno + " out of range");
   raf.seek(recno * RECLEN);
   write(partnum, partdesc, qty, ucost);
}

private Part read() throws IOException
{
   StringBuffer sb = new StringBuffer();
   for (int i = 0; i < PNUMLEN; i++)
      sb.append(raf.readChar());
   String partnum = sb.toString().trim();
   sb.setLength(0);
   for (int i = 0; i < DESCLEN; i++)
      sb.append(raf.readChar());
   String partdesc = sb.toString().trim();
   int qty = raf.readInt();
   int ucost = raf.readInt();
   return new Part(partnum, partdesc, qty, ucost);
}
```

Реалізація частин БД на плоских файлах

- Метод numRecs() повертає кількість записів у файлі.
  - Починаючи з 0 та закінчуючи numRecs() - 1.
  - Методи select() та update() перевіряють, щоб їх аргумент recno знаходився в цьому діапазоні.
- Метод select() викликає приватний метод read(), щоб повернути запис з ID recno як екземпляр вкладеного класу Part.
  - Конструктор класу Part ініціалізує об’єкт Part значеннями полів із запису, а його геттери повертають ці значення.
- Метод update() спочатку встановлює file pointer на початок запису (recno).
  - Як і в методі append(), викликається write() для виводу його аргументів, проте замість додавання запису він заіняється.

```
private void write(String partnum, String partdesc, int qty, int ucost)
   throws IOException
{
   StringBuffer sb = new StringBuffer(partnum);
   if (sb.length() > PNUMLEN)
      sb.setLength(PNUMLEN);
   else
   if (sb.length() < PNUMLEN)
   {
      int len = PNUMLEN - sb.length();
      for (int i = 0; i < len; i++)
         sb.append(" ");
   }
   raf.writeChars(sb.toString());
   sb = new StringBuffer(partdesc);
   if (sb.length() > DESCLEN)
      sb.setLength(DESCLEN);
   else
   if (sb.length() < DESCLEN)
   {
      int len = DESCLEN - sb.length();
      for (int i = 0; i < len; i++)
         sb.append(" ");
   }
   raf.writeChars(sb.toString());
   raf.writeInt(qty);
   raf.writeInt(ucost);
}
```

- Записування записів відбувається за допомогою приватного методу write().
  - Оскільки поля повинні мати точний розмір, write() «роздуває» коротші за розмір поля рядкові значення пробілами справа та видаляє ці значення за потреби.
- Записи зчитуються за допомогою приватного методу read(), який видаляє «роздування» до того, як зберігати рядкове значення поля в об’єкті Part.

```
public static class Part
{
   private String partnum;
   private String desc;
   private int qty;
   private int ucost;

   public Part(String partnum, String desc, int qty, int ucost)
   {
      this.partnum = partnum;
      this.desc = desc;
      this.qty = qty;
      this.ucost = ucost;
   }

   String getDesc()
   {
      return desc;
   }

   String getPartnum()
   {
      return partnum;
   }
   int getQty()
   {
      return qty;
   }

   int getUnitCost()
   {
      return ucost;
   }
}
}
```

- Сам по собі клас PartsDB цінності не має.
  - Потрібен додаток, що дозволить працювати з ним.

```
import java.io.IOException;

public class UsePartsDB
{
   public static void main(String[] args)
   {
      PartsDB pdb = null;
      try
      {
         pdb = new PartsDB("parts.db");
         if (pdb.numRecs() == 0)
         {
            // Populate the database with records.
            pdb.append("1-9009-3323-4x", "Wiper Blade Micro Edge", 30, 2468);
            pdb.append("1-3233-44923-7j", "Parking Brake Cable", 5, 1439);
            pdb.append("2-3399-6693-2m", "Halogen Bulb H4 55/60W", 22, 813);
            pdb.append("2-599-2029-6k", "Turbo Oil Line O-Ring ", 26, 155);
            pdb.append("3-1299-3299-9u", "Air Pump Electric", 9, 20200);
         }
         dumpRecords(pdb);
         pdb.update(1, "1-3233-44923-7j", "Parking Brake Cable", 5, 1995);
         dumpRecords(pdb);
      }
      catch (IOException ioe)
      {
         System.err.println(ioe);
      }
      finally
      {
         if (pdb != null)
            pdb.close();
      }
   }
```

Експериментуємо з частинами БД

- Зауважте, що значення цін зберігаються в цілочисельних кількостях пенні.
  - Наприклад, задається літерал 1995 для представлення $19.95.
  - Якби використовувались об’єкти java.math.BigDecimal для зберігання грошових значень, потрібно було б виконати рефакторинг PartsDB з урахуванням серіалізації (яка ще не розглядалась)

```
static void dumpRecords(PartsDB pdb) throws IOException
{
   for (int i = 0; i < pdb.numRecs(); i++)
   {
      PartsDB.Part part = pdb.select(i);
      System.out.print(format(part.getPartnum(), PartsDB.PNUMLEN, true));
      System.out.print(" | ");
      System.out.print(format(part.getDesc(), PartsDB.DESCLEN, true));
      System.out.print(" | ");
      System.out.print(format("" + part.getQty(), 10, false));
      System.out.print(" | ");
      String s = part.getUnitCost() / 100 + "." + part.getUnitCost() % 100;
      if (s.charAt(s.length() - 2) == '.') s += "0";
       System.out.println(format(s, 10, false));
   }
   System.out.println("Number of records = " + pdb.numRecs());
   System.out.println();
}
```

```
static String format(String value, int maxWidth, boolean leftAlign)
{
   StringBuffer sb = new StringBuffer();
   int len = value.length();
   if (len > maxWidth)
   {
      len = maxWidth;
      value = value.substring(0, len);
   }
   if (leftAlign)
   {
      sb.append(value);
      for (int i = 0; i < maxWidth-len; i++)
         sb.append(" ");
   }
   else
   {
      for (int i = 0; i < maxWidth-len; i++)
         sb.append(" ");
      sb.append(value);
   }
   return sb.toString();
}
}
```

- main() використовує допоміжний метод dumpRecords(), щоб dump записи, а dumpRecords() покладається на допоміжний метод format(), який форматує значення полів для їх представлення в коректно вирівняних стовпцях
  - Замість нього можна було використовувати java.util.Formatter.

Програмний вивід

```
1-9009-3323-4x      | Wiper Blade Micro Edge      |    30 |    24.68
1-3233-44923-7j     | Parking Brake Cable         |     5 |    14.39
2-3399-6693-2m      | Halogen Bulb H4 55/60W      |    22 |     8.13
2-599-2029-6k       | Turbo Oil Line O-Ring       |    26 |     1.55
3-1299-3299-9u      | Air Pump Electric           |     9 |   202.00
Number of records = 5

1-9009-3323-4x      | Wiper Blade Micro Edge      |    30 |    24.68
1-3233-44923-7j     | Parking Brake Cable         |     5 |    19.95
2-3399-6693-2m      | Halogen Bulb H4 55/60W      |    22 |     8.13
2-599-2029-6k       | Turbo Oil Line O-Ring       |    26 |     1.55
3-1299-3299-9u      | Air Pump Electric           |     9 |   202.00
Number of records = 5
```