برنامه نویسی پیشرفته

وحیدی اصل

برنامه نویسی ورودی/خروجی (I/O)

Java I/O

- در این درس نگاهی جامع به مفهوم ورودی/خروجی (I/O) و کلاسهای مربوطه در پکیج (java.io) خواهیم داشت.
- کلاسهای مختلف موجود در این پکیج را برمبنای هدف و کاربرد آنها دسته بندی می کنیم.
- با تسلط یافتن بر مفاهیم این درس و گروهبندی ارایه شده قادر خواهید بود:
  - برای یک کاربرد و هدف مشخص ورودی/خروجی، مناسبترین کلاس را برگزینید.
  - با مشاهده یکی از این کلاسها در برنامه به هدف آن برنامه پی ببرید.

ورودی و خروجی- مبدا و مقصد

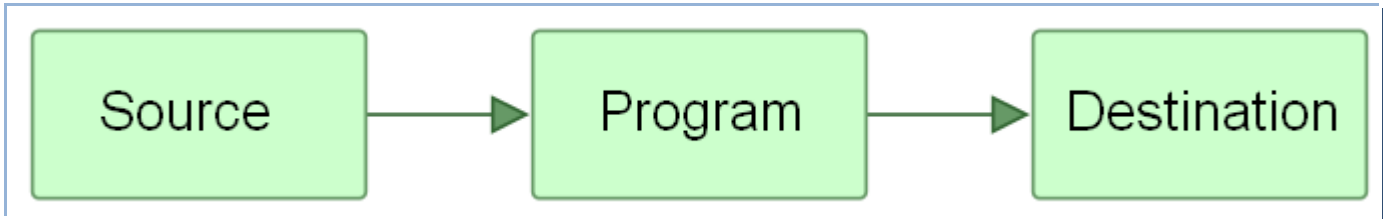
- کلمات ورودی و خروجی برخی اوقات ممکن است مبهم باشند.
- ورودی بخشی از یک برنامه می‌تواند خروجی بخشی دیگر باشد.
- آیا یک OutputStream جریانی است که خروجی را در آنجا می نویسیم یا از آن داده‌ها را می‌خوانیم (داده‌هایی برای خواندن از آن خارج می‌شود) و به همین سبب OutputStream نام گرفته است؟
- آیا یک InputStream جریانی است که برنامه شما داده‌ها را از آن می‌خواند؟
- در این درس، تفاوت دقیق این مفاهیم آشکار می‌شوند.

پکیج IO در جاوا

- هدف از طراحی کلاسهایی در پکیج IO، خواندن داده‌هایی از یک مبدا (منبع) و نوشتن داده‌ها در یک مقصد است.
- متداولترین مبداها (منابع) و مقصدهای داده‌ای در زیر نشان داده شده‌اند:

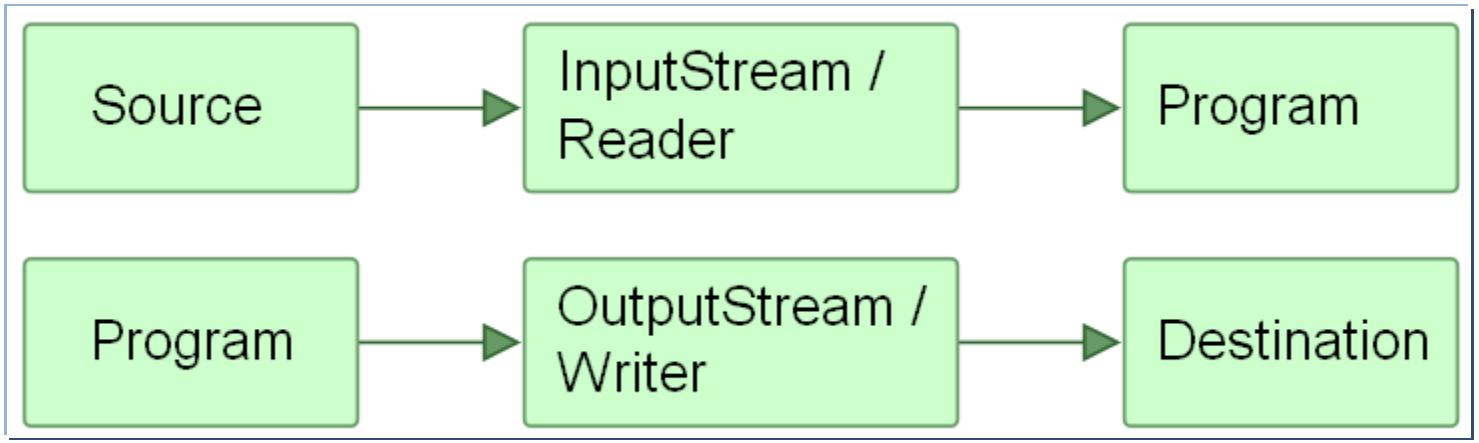
- Files
- Pipes
- Network Connections
- In-memory Buffers (e.g. arrays)
- System.in, System.out, System.error

- نمودار زیر نشان می‌دهد یک برنامه جاوا چگونه داده‌ها را از یک مبدا می‌خواند و در یک مقصد می‌نویسد.



InputStream, OutputStream, Reader and Writer

- اگر برنامه بخواهد داده‌ای را از یک منبع بخواند، به یک کلاس InputStream یا Reader نیاز دارد.
- اگر برنامه بخواهد داده‌ای را در یک مقصد بنویسد، به یک کلاس OutputStream یا Writer نیاز دارد.
- یک InputStream یا Reader به یک منبع داده‌ای متصل می‌شود.
- یک OutputStream یا Writer به یک مقصد داده‌ای متصل می‌شود.



اهداف و خصوصیات Java IO

- پکیج Java IO حاوی تعداد زیادی زیرکلاس برای کلاسهای InputStream، Reader، OutputStream و Writer است.
- دلیل این تعدد زیرکلاسها آن است که هریک، هدف خاص و متفاوتی را دنبال می‌کنند.
- برخی از اهداف تعریف زیرکلاسهای متعدد:

- File Access
- Network Access
- Internal Memory Buffer Access
- Inter-Thread Communication (Pipes)
- Buffering
- Filtering
- Parsing
- Reading and Writing Text (Readers / Writers)
- Reading and Writing Primitive Data (long, int etc.)
- Reading and Writing Objects

Java IO Class Overview Table

	Byte Based		Character Based	
	Input	Output	Input	Output
Basic	InputStream	OutputStream	Reader InputStreamReader	Writer OutputStreamWriter
Arrays	ByteArrayInputStream	ByteArrayOutputStream	CharArrayReader	CharArrayWriter
Files	FileInputStream RandomAccessFile	FileOutputStream RandomAccessFile	FileReader	FileWriter
Pipes	PipedInputStream	PipedOutputStream	PipedReader	PipedWriter
Buffering	BufferedInputStream	BufferedOutputStream	BufferedReader	BufferedWriter
Filtering	FilterInputStream	FilterOutputStream	FilterReader	FilterWriter
Parsing	PushbackInputStream StreamTokenizer		PushbackReader LineNumberReader	
Strings			StringReader	StringWriter
Data	DataInputStream	DataOutputStream		
Data - Formatted		PrintStream		PrintWriter
Objects	ObjectInputStream	ObjectOutputStream		
Utilities	SequenceInputStream			

کلاس File

- با استفاده از کلاس فایل در Java IO API می‌توانیم به سیستم فایل کامپیوتر دسترسی پیدا کنیم.
- کلاس فایل به شما امکان می‌دهد:
  - بررسی کنید آیا یک فایل یا دایرکتوری (فهرست) وجود دارد؟
  - فایل یا مسیرفایل را در قالب نام فایل که یک رشته است، ایجاد کنید.
  - اطلاعات مربوط به فایل مانند وجود فایل (دایرکتوری)، قابلیت خواندن یا نوشتن آن، طول آن، مسیر آن، زمان تغییر، نام آن، پنهان بودن آن، را کسب نماییم .
  - لیستی از فایلها درون یک دایرکتوری را بخوانیم.
  - همچنین می‌توانیم فایل را پاک کنیم یا نامش را تغییر دهیم.
  - در اسلایدهای بعدی برخی از متدهای کلاس File که برروی اشیای حاصل از این کلاس قابل اعمال هستند، نشان داده شده‌اند.
- توجه کنید که کلاس فایل به شما امکان خواندن از یا نوشتن در فایلها را نمی‌دهد.
- برای خواندن/نوشتن باید از کلاسهای FileInputStream، FileOutputStream و RandomAccessFile استفاده کنید.

کسب اطلاعات از خصوصیات فایل یا دستکاری آن

java.io.File	
+File(pathname: String)	Creates a File object for the specified pathname. The pathname may be a directory or a file.
+File(parent: String, child: String)	Creates a File object for the child under the directory parent. child may be a filename or a subdirectory.
+File(parent: File, child: String)	Creates a File object for the child under the directory parent. parent is a File object. In the preceding constructor, the parent is a string.
+exists(): boolean	Returns true if the file or the directory represented by the File object exists.
+canRead(): boolean	Returns true if the file represented by the File object exists and can be read.
+canWrite(): boolean	Returns true if the file represented by the File object exists and can be written.
+isDirectory(): boolean	Returns true if the File object represents a directory.
+isFile(): boolean	Returns true if the File object represents a file.
+isAbsolute(): boolean	Returns true if the File object is created using an absolute path name.
+isHidden(): boolean	Returns true if the file represented in the File object is hidden. The exact definition of *hidden* is system-dependent. On Windows, you can mark a file hidden in the File Properties dialog box. On Unix systems, a file is hidden if its name begins with a period character '.'.
+getAbsolutePath(): String	Returns the complete absolute file or directory name represented by the File object.
+getCanonicalPath(): String	Returns the same as getAbsolutePath() except that it removes redundant names, such as "." and "..", from the pathname, resolves symbolic links (on Unix platforms), and converts drive letters to standard uppercase (on Win32 platforms).
+getName(): String	Returns the last name of the complete directory and file name represented by the File object. For example, new File("c:\\book\\test.dat").getName() returns test.dat.
+getPath(): String	Returns the complete directory and file name represented by the File object. For example, new File("c:\\book\\test.dat").getPath() returns c:\book\test.dat.
+getParent(): String	Returns the complete parent directory of the current directory or the file represented by the File object. For example, new File("c:\\book\\test.dat").getParent() returns c:\book.
+lastModified(): long	Returns the time that the file was last modified.
+delete(): boolean	Deletes this file. The method returns true if the deletion succeeds.
+renameTo(dest: File): boolean	Renames this file. The method returns true if the operation succeeds.

ایجاد یک شیئ از کلاس java.io.File

- قبل از آنکه بخواهید با یک سیستم فایل کار کنید یا از امکانات کلاس فایل استفاده کنید باید ابتدا یک نمونه (شیئ) از کلاس File ایجاد کنید.
- برای مثال:

```
file = new File(“e:\\data\\input-file.txt");
```

- این یک نمونه از سازنده های کلاس File است. این کلاس سازنده‌های مشابهی نیز دارد که بسته به کاربردتان می‌توانید از هریک استفاده کنید.
- شیئ file مسیر "c:\\data\\input-file.txt" را در قالب یک رشته نگهداری می‌کند و از این پس قادر است عملیاتی را بروی این مسیر (فایل یا دایرکتوری) انجام دهد.

بررسی وجود یک فایل

- پس از ایجاد یک شیئ از کلاس فایل می‌توانید بررسی کنید آیا فایل مشخص‌شده در رشته حاوی مسیر در شیئ، وجود خارجی دارد یا خیر.
- سازنده کلاس File حتی اگر چنین فایلی در مسیرمشخص‌شده وجود نداشته باشد، دچار اشکال نمی‌شود.
- اگر چنین فایلی وجود نداشته باشد، می‌توانید آن را ایجاد کنید.
- مثال زیر را ببینید:

```
File file = new File("c:\\data\\input-file.txt");
boolean fileExists = file.exists();
```

- این متد اگر چنین مسیری وجود داشته باشد، true بر می‌گرداند. اگر فایل مربوطه در مسیر مشخص شده وجود نداشته باشد، با دستور زیر آن را ایجاد کنید.

```
file.createNewFile();
```

- برای ایجاد دایرکتوریها از دستورات mkdir() و mkdirs() در اسلاید بعد استفاده کنید.

ایجاد یک دایرکتوری اگر موجود نباشد

- با استفاده از کلاس File می‌توانید درصورت عدم وجود دایرکتوری مربوطه، دایرکتوری ایجاد کنید.
- کلاس File دو متد mkdir () و mkdirs() دارد که برای شما طبق مسیری که مشخص کرده‌اید، دایرکتوری می‌سازند.
- متد mkdir() در صورت عدم وجود دایرکتوری، یک دایرکتوری جدید می‌سازد.
- مثال:

```
File file = new File("c:\\users\\AP\\Java");
 boolean dirCreated = file.mkdir();
```

بافرض آنکه دایرکتوری c:\users\AP از قبل موجود باشد، کد بالا یک زیردایرکتوری به نام Java می سازد.

- mkdir() مقدار true برمی‌گرداند اگر دایرکتوری ایجاد شود و در غیراینصورت false برمی‌گرداند.
- اگر بخواهید کل دایرکتوری را بسازید (c:\users\AP از قبل موجود نباشد) از متد mkdirs() استفاده کنید.

```
File file = new File("c:\\users\\AP\\newdir");
 boolean dirCreated = file.mkdirs();
```

اندازه فایل

- با متد length() می‌توانید از اندازه فایل دلخواه برحسب بایت آگاه شوید.
- مثال:

```
File file = new File("c:\\data\\input-file.txt");
 long length = file.length();
```

تغییرنام یا جابجایی فایل

- برای تغییرنام یا جابجایی یک فایل از متد renameTo() استفاده کنید.
- مثال:

- File file = new File("c:\\data\\input-file.txt");
- boolean success = file.renameTo(new File("c:\\data\\new-file.txt"));

- با این متد همچنین می‌توانید یک فایل را به یک دایرکتوری جدید منتقل کنید.
- لزومی ندارد که نام فایل جدید در همان مسیرفایل قبلی باشد.
- اگر عمل تغییرنام یا جابجایی موفقیت‌آمیز بود، مقدار true برگردانده خواهد شد و در غیراینصورت false برمی‌گردد. (مثلا اگر فایل باز باشد)

پاک کردن فایل

- برای حذف فایل از متد delete() استفاده می‌کنیم.
- مثال:

```
File file = new File("c:\\data\\input-file.txt");
boolean success = file.delete();
```

- بسته به موفقیت آمیز بودن یا نبودن این عملیات به ترتیب مقادیر true یا false برمی‌گردد.

بررسی اینکه یک مسیر، فایل است یا دایرکتوری

- شیئی از کلاس فایل، می‌تواند هم یک فایل و هم یک دایرکتوری را کنترل کند.
- برای تشخیص این موضوع، می‌توانید از متد isDirectory() یا isFile() استفاده کنید:

```
File file = new File("c:\\data");
boolean isDirectory = file.isDirectory();  //true
```

خواندن لیستی از فایلها در دایرکتوری

- برای دانستن لیست پوشه‌ها و فایلهای موجود در یک دایرکتوری از متد list() یا listFiles() استفاده می‌کنیم.
- خروجی این متدها آرایه‌ای از نوع String می‌باشد.
- مسیری که می‌خواهیم محتویات آن را بازیابی کنیم، باید از جنس دایرکتوری باشد.
- مثال:

- File file = new File("c:\\data");
- String[] fileNames = file.list();
  - محتویات دایرکتوری مشخص شده را در یک آرایه از رشته‌ها ذخیره‌سازی می‌کند.
- File[] files = file.listFiles();
- محتویات دایرکتوری مشخص‌شده را در یک آرایه از فایلها (از نوع File کلاس ریموت کنترل) ذخیره‌سازی می‌کند.

مثالی از کلاس فایل

```
public class TestFileClass {
  public static void main(String[] args) throws IOException{
    java.io.File file = new java.io.File(" C:\\Test\\test.txt ");

     file.createNewFile();
    System.out.println("Does it exist? " + file.exists());
    System.out.println("The file has " + file.length() + " bytes");
    System.out.println("Can it be read? " + file.canRead());
    System.out.println("Can it be written? " + file.canWrite());
    System.out.println("Is it a directory? " + file.isDirectory());
    System.out.println("Is it a file? " + file.isFile());
    System.out.println("Is it absolute? " + file.isAbsolute());
    System.out.println("Is it hidden? " + file.isHidden());
    System.out.println("Absolute path is " +    file.getAbsolutePath());
    System.out.println("Last modified on " +   new java.util.Date(file.lastModified()));
  }
}
```

Text I/O

- یک شیئ File نام یک فایل یا یک مسیر فایل را ذخیره می‌کند.
- اما دارای متدهایی جهت خواندن/نوشتن داده ها از/به فایل نمی‌باشد.

- برای انجام عملیات ورودی/خروجی (I/O) باید از کلاسهایی استفاده کنیم که حاوی متدهایی برای خواندن/نوشتن از/به فایل باشند.

- در ادامه می‌آموزیم چگونه رشته‌ها و مقادیر عددی را با استفاده از کلاسهای Scanner و PrintWriter از/به فایلها بنویسیم/بخوانیم.

نوشتن داده ها با استفاده از PrintWriter

- کلاس PrintWriter به شما امکان می‌دهد داده‌های دارای ساختار (فرمت) را در یک فایل مشخص بنویسید.
- برای مثال می‌توانید مقادیر int، long، و سایر داده‌های اصلی را در قالب یک متن بنویسید.
- یک مثال ساده:

```
PrintWriter writer = new PrintWriter(file1);
writer.print(true);
writer.print((int) 123);
writer.print((float) 123.456);
writer.printf("Text + data: %d", 123);
 writer.close();
```

- این کلاس دارای متدهای printf() , format() می‌باشد.
- با استفاده از متد print در این کلاسها می‌توانید داده‌ها را در قالب رشته یا آرایه‌ای از کاراکترها در فایل بنویسید.
- این کلاس سازندهای متنوعی دارد که به شما امکان می‌دهد به یک File، یک OutputStream یا یک Writer متصل شوید.

نوشتن داده ها با استفاده از PrintWriter

java.io.PrintWriter	
+PrintWriter(filename: String)	Creates a PrintWriter for the specified file.
+print(s: String): void	Writes a string.
+print(c: char): void	Writes a character.
+print(cArray: char[]): void	Writes an array of character.
+print(i: int): void	Writes an int value.
+print(l: long): void	Writes a long value.
+print(f: float): void	Writes a float value.
+print(d: double): void	Writes a double value.
+print(b: boolean): void	Writes a boolean value.
Also contains the overloaded println methods.	A println method acts like a print method; additionally it prints a line separator. The line separator string is defined by the system. It is \r\n on Windows and \n on Unix.
Also contains the overloaded printf methods.	The printf method was introduced in §3.6, "Formatting Console Output and Strings."

نوشتن داده ها با استفاده از PrintWriter

```
public class WriteData {
  public static void main(String[] args) throws Exception {
    java.io.File file = new java.io.File("scores.txt");

    // Create a file
    java.io.PrintWriter output = new java.io.PrintWriter(file);

    // Write formatted output to the file
    output.print("John T Smith ");
    output.println(90);
    output.print("Eric K Jones ");
    output.println(85);

    // Close the file
    output.close();
  }
}
```

خواندن داده ها با استفاده از Scanner

java.util.Scanner	
+Scanner(source: File)	Creates a Scanner that produces values scanned from the specified file.
+Scanner(source: String)	Creates a Scanner that produces values scanned from the specified string.
+close()	Closes this scanner.
+hasNext(): boolean	Returns true if this scanner has another token in its input.
+next(): String	Returns next token as a string.
+nextByte(): byte	Returns next token as a byte.
+nextShort(): short	Returns next token as a short.
+nextInt(): int	Returns next token as an int.
+nextLong(): long	Returns next token as a long.
+nextFloat(): float	Returns next token as a float.
+nextDouble(): double	Returns next token as a double.
+useDelimiter(pattern: String): Scanner	Sets this scanner's delimiting pattern.

خواندن داده ها با استفاده از Scanner

```
import java.util.Scanner;

public class ReadData {
  public static void main(String[] args) throws Exception {
    // Create a File instance
    java.io.File file = new java.io.File("scores.txt");

    // Create a Scanner for the file
    Scanner input = new Scanner(file);

    // Read data from a file
    while (input.hasNext()) {
      String firstName = input.next();
      String mi = input.next();
      String lastName = input.next();
      int score = input.nextInt();
      System.out.println(
        firstName + " " + mi + " " + lastName + " " + score);
    }
    // Close the file
    input.close();
  }
}
```

کلاس FileWriter

- کلاس FileWriter برای نوشتن داده‌های کاراکتری درون یک فایل استفاده می‌شود. شرکت میکروسیستم Sun پیشنهاد کرده در مواردی که فایلهای مورد استفاده، متنی هستند از همین کلاس و کلاس FileReader به ترتیب برای نوشتن و خواندن داده‌ها استفاده شود و کلاسهای FileInputStream و FileOutputStream مورد استفاده قرار نگیرند.

```
import java.io.*;
class Simple{
 public static void main(String args[]){
  try{
   FileWriter fw=new FileWriter("abc.txt");
   fw.write("my name is sachin");
   fw.flush();

   fw.close();
  }catch(Exception e){System.out.println(e);}
  System.out.println("success");
 }
}
```

```
Output:success...
```

کلاس FileReader

- کلاس FileReader برای خواندن داده‌های کاراکتری از یک فایل استفاده می‌شود.
- متد read() در این کلاس یک مقدار int برمی‌گرداند که حاوی مقدار کاراکتری کاراکتر خوانده‌شده است. اگر read() مقدار 1- را برگرداند، یعنی داده بیشتَری در FileReader برای خواندن وجود ندارد و می‌توانیم آن را close() کنیم.

```
import java.io.*;
class Simple{
 public static void main(String args[]) {

  FileReader fr=new FileReader("abc.txt");
  int i;
  while((i=fr.read())!=-1)
  System.out.println((char)i);

  fr.close();
 }
}
```

```
Output:my name is sachin
```