5 Text-Based Applications

Topics

- Command-Line Arguments and System Properties
- Reading from Standard Input
- File Handling
  - Reading from a File
  - Writing to a File

Command-Line Arguments

- Java allows user to input data from the command line
  - Purpose of declaring *String args[]* as a parameter in the main method
  - When using the *java* command, specifying data after the class name indicates you are passing data via the args parameter

- Example:

```
java Calculate 1 2
```

  - *args[0]* has the value *"1"*
  - *args[1]* has the value *"2"*

System Properties

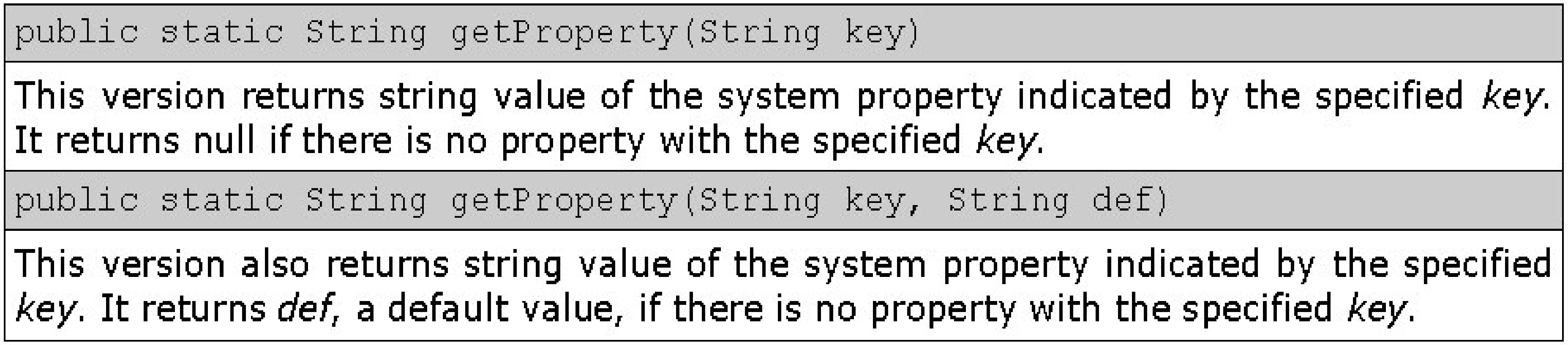
- Java also also allows you to manipulate system properties from the command line

- System property
  - Quite similar to environment variables
  - But is not platform-dependent

- Property
  - Mapping between the property name to its corresponding value
  - Represented in Java with the *Properties* class.

System Properties

- *System* class
  - Provides a methods for determining the current system properties, the *getProperties* method that returns a *Properties* object
  - Also provides the overloaded *getProperty* method



`public static String getProperty(String key)`
This version returns string value of the system property indicated by the specified *key*. It returns null if there is no property with the specified *key*.
`public static String getProperty(String key, String def)`
This version also returns string value of the system property indicated by the specified *key*. It returns *def*, a default value, if there is no property with the specified *key*.

System Properties

- Including a new property
  - Use the *-D* option with the *java* command

```
java -D<name>=value
```

  - Example:

```
java -Duser.home=philippines
```

- Display the list of system properties
  - Use the *getProperties* method

```
System.getProperties().list(System.out);
```

Reading from Standard Input: Streams

- Can use streams to read from standard input
- Stream
  - Abstraction of a file or a device that allows a series of items to be read or written
  - Connected to physical devices
  - Two general kinds of streams:
    - Character streams
    - Byte streams

Reading from Standard Input: Streams

- Character Streams
  - For Unicode characters
- Byte Streams
  - For binary data
  - Predefined examples
    - *System.in* (keyboard by default)
    - *System.out* (console by default)

Reading from Standard Input: *BufferedReader*

- Reading characters from the keyboard
  - Use the *System.in* byte stream warped in a *BufferedReader* object

```
BufferedReader br = new BufferedReader(new
                  InputStreamReader(System.in));
```

  - Use *read* method of the *BufferedReader* object

```
ch = (int) br.read();
//read method returns an integer
```

Reading from Standard Input: *BufferedReader* Example

```
import java.io.*;
class FavoriteCharacter {
   public static void main(String args[])
                                throws IOException {
      System.out.println("Hi, what’s your favorite
                               character?");
      char favChar;
      BufferedReader br = new BufferedReader(new
                     InputStreamReader(System.in));
      favChar = (char) br.read();
      System.out.println(favChar +
                     " is a good choice!");
   }
}
```

Reading from Standard Input: *BufferedReader*

- Reading an entire line
  - Use the *System.in* byte stream warped in a *BufferedReader* object

```
BufferedReader br = new BufferedReader(new
                 InputStreamReader(System.in));
```

  - Use the *readLine* method

```
str = br.readLine();
```

Reading from Standard Input: *BufferedReader* Example

```java
import java.io.*;
class GreetUser {
   public static void main(String args[])
                                throws IOException {
      System.out.println("Hi, what's your name?");
      String name;
      BufferedReader br = new BufferedReader(new
                    InputStreamReader(System.in));
      name = br.readLine();
      System.out.println("Nice to meet you, " +
                           name + "! :)");
   }
}
```

Reading from Standard Input: Reminders

- Don’t forget to import the *java.io* package as shown below:

```
import java.io.*;
```

- Reading from streams may cause checked exceptions to occur
  - Handle these exceptions using *try-catch* statements
  - Or handle by indicating the exception in the throws clause of the method

File Handling: Reading from a File

- Can use the *FileInputStream* class
  - One of the constructors of this class

    ```
    FileInputStream(String filename)
    ```

  - Creates a connection to an actual file whose *filename* is specified as an argument
  - A *FileNotFoundException* is thrown when the file does not exist or it cannot be opened for reading

- Using the read method
  - Returns an integer representation of data read
  - Returns -1 when the end of the file is reached

File Handling: Reading from a File

```
import java.io.*;
class ReadFile {
   public static void main(String args[])
                                   throws IOException {
      System.out.println("What is the name of the
                                 file to read from?");
      String filename;
      BufferedReader br = new BufferedReader(new
                           InputStreamReader(System.in));
      filename = br.readLine();
      System.out.println("Now reading from " +
                                 filename + "...");
//continued...
```

File Handling: Reading from a File

```
        FileInputStream fis = null;
        try {
            fis = new FileInputStream(filename);
        } catch (FileNotFoundException ex) {
            System.out.println("File not found.");
        }
        try {
            char data;
            int temp;
//continued...
```

File Handling: Reading from a File

```
            do {
                temp = fis.read();
                data = (char) temp;
                if (temp != -1) {
                    System.out.print(data);
                }
            } while (temp != -1);
        } catch (IOException ex) {
            System.out.println("Problem in reading from
                                          the file.");
        }
    }
}
```

File Handling: Writing to a File

- Can use the *FileOutputStream* class
  - One of the constructors of this class

    ```
    FileOutputStream(String filename)
    ```

  - Links an output stream to an actual file to write to
  - A *FileNotFoundException* is thrown when the file cannot be opened for writing

- Using the *write* method

  ```
  void write(int b)
  ```

  where,

  - *b* refers to the data to be written to the actual file

File Handling: Writing to a File

```java
import java.io.*;
class WriteFile {
   public static void main(String args[])
                                    throws IOException {
      System.out.println("What is the name of the
                                   file to be written to?");
      String filename;
      BufferedReader br = new BufferedReader(new
                              InputStreamReader(System.in));
      filename = br.readLine();
      System.out.println("Enter data to write to " +
                                  filename + "...");
//continued...
```

File Handling: Writing to a File

```
      System.out.println("Type q$ to end.");
      FileOutputStream fos = null;
      try {
         fos = new FileOutputStream(filename);
      } catch (FileNotFoundException ex) {
         System.out.println("File cannot be opened
                                   for writing.");
      }
      try {
         boolean done = false;
         int data;
//continued...
```

File Handling: Writing to a File

```
    do {
        data = br.read();
        if ((char)data == ’q’) {
            data = br.read();
            if ((char)data == ’$’) {
                done = true;
            } else {
                fos.write(’q’);
                fos.write(data);
            }
        } else {
            fos.write(data);
//continued...
```

File Handling: Writing to a File

```
            }
         } while (!done);
  } catch (IOException ex) {
         System.out.println("Problem in reading from
                             the file.");
      }
   }
}
```

Summary

- Command-Line Arguments and System Properties
  - Getting input from the command line
  - Manipulating system properties

```
java -D<name>=<value>
```

- Reading from Standard Input
  - Use *System.in*
  - Use *BufferedReader*
  - Use *read* method

Summary

- File Handling
  - Reading from a File
    - Use *FileInputStream*
    - Use *read* method
  - Writing to a File
    - Use *FileOutputStream*
    - Use *write* method