Trail: Essential Classes
Lesson: Basic I/O
Section: I/O Streams

The Java Tutorials have been written for JDK 8. Examples and practices described in this page don't take advantage of improvements introduced in later releases.

I/O from the Command Line

A program is often run from the command line and interacts with the user in the command line environment. The Java platform supports this kind of interaction in two ways: through the Standard Streams and through the Console.

Standard Streams

Standard Streams are a feature of many operating systems. By default, they read input from the keyboard and write output to the display. They also support I/O on files and between programs, but that feature is controlled by the command line interpreter, not the program.

The Java platform supports three Standard Streams: *Standard Input*, accessed through `System.in`; *Standard Output*, accessed through `System.out`; and *Standard Error*, accessed through `System.err`. These objects are defined automatically and do not need to be opened. Standard Output and Standard Error are both for output; having error output separately allows the user to divert regular output to a file and still be able to read error messages. For more information, refer to the documentation for your command line interpreter.

You might expect the Standard Streams to be character streams, but, for historical reasons, they are byte streams. `System.out` and `System.err` are defined as `PrintStream` objects. Although it is technically a byte stream, `PrintStream` utilizes an internal character stream object to emulate many of the features of character streams.

By contrast, `System.in` is a byte stream with no character stream features. To use Standard Input as a character stream, wrap `System.in` in `InputStreamReader`.

```
InputStreamReader cin = new InputStreamReader(System.in);
```

The Console

A more advanced alternative to the Standard Streams is the Console. This is a single, predefined object of type `Console` that has most of the features provided by the Standard Streams, and others besides. The Console is particularly useful for secure password entry. The Console object also provides input and output streams that are true character streams, through its `reader` and `writer` methods.

Before a program can use the Console, it must attempt to retrieve the Console object by invoking `System.console()`. If the Console object is available, this method returns it. If `System.console` returns `NULL`, then Console operations are not permitted, either because the OS doesn't support them or because the program was launched in a noninteractive environment.

The Console object supports secure password entry through its `readPassword` method. This method helps secure password entry in two ways. First, it suppresses echoing, so the password is not visible on the user's screen. Second, `readPassword` returns a character array, not a `String`, so the password can be overwritten, removing it from memory as soon as it is no longer needed.

The `Password` example is a prototype program for changing a user's password. It demonstrates several `Console` methods.

```
import java.io.Console;
import java.util.Arrays;
import java.io.IOException;

public class Password {

    public static void main (String args[]) throws IOException {

        Console c = System.console();
        if (c == null) {
            System.err.println("No console.");
            System.exit(1);
        }

        String login = c.readLine("Enter your login: ");
```

```
            char [] oldPassword = c.readPassword("Enter your old password: ");

            if (verify(login, oldPassword)) {
                boolean noMatch;
                do {
                    char [] newPassword1 = c.readPassword("Enter your new password: ");
                    char [] newPassword2 = c.readPassword("Enter new password again: ");
                    noMatch = ! Arrays.equals(newPassword1, newPassword2);
                    if (noMatch) {
                        c.format("Passwords don't match. Try again.%n");
                    } else {
                        change(login, newPassword1);
                        c.format("Password for %s changed.%n", login);
                    }
                    Arrays.fill(newPassword1, ' ');
                    Arrays.fill(newPassword2, ' ');
                } while (noMatch);
            }

            Arrays.fill(oldPassword, ' ');
        }

        // Dummy change method.
        static boolean verify(String login, char[] password) {
            // This method always returns
            // true in this example.
            // Modify this method to verify
            // password according to your rules.
            return true;
        }

        // Dummy change method.
        static void change(String login, char[] password) {
            // Modify this method to change
            // password according to your rules.
        }
    }
```

The `Password` class follows these steps:

1. Attempt to retrieve the Console object. If the object is not available, abort.
2. Invoke `Console.readLine` to prompt for and read the user's login name.
3. Invoke `Console.readPassword` to prompt for and read the user's existing password.
4. Invoke `verify` to confirm that the user is authorized to change the password. (In this example, `verify` is a dummy method that always returns `true`.)
5. Repeat the following steps until the user enters the same password twice:
    a. Invoke `Console.readPassword` twice to prompt for and read a new password.
    b. If the user entered the same password both times, invoke `change` to change it. (Again, `change` is a dummy method.)
    c. Overwrite both passwords with blanks.
6. Overwrite the old password with blanks.

Previous page: Formatting
Next page: Data Streams