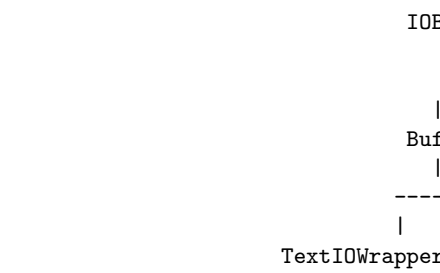
Tester avec des entrées-sorties

Master info S2
TD SVL – 2017-2018

Comment tester une application qui utilise des flots d'entrées-sorties type clavier, console, fichiers, etc. (Le test d'une interface graphique est un problème en soi.)

Méthode générale Dans des tests automatisés, pas question d'entrer une valeur au clavier ou d'aller vérifier le contenu d'un fichier qu'on a écrit. On remplace les flots applicatifs par des flots utilisés uniquement pour les tests, contenant les données à lire (flot d'entrée) ou recevant les données à écrire (flot de sortie). Pour ce faire, l'idée est comme d'habitude d'injecter les dépendances dans le constructeur ou les méthodes de la classe à tester. La classe à tester dépend donc explicitement des flots d'entrée et de sortie, qui seront des super-types des types utilisés en pratique dans les tests ou dans le code applicatif. L'idée est qu'on peut utiliser dans les tests des objets d'un type fournis par le langage, par exemple `io.StringIO` pour Python ou `ByteArrayInputStream` en Java, qu'on peut facilement manipuler via des chaînes de caractère (`str` pour Python et `String` pour Java). Pour les entrées : on construit une chaîne qui est transformée en flot d'entrée. Pour les sorties : on écrit dans un flot de sortie et on sait le transformer en chaîne pour vérifier ce qu'on a écrit. On évite ainsi d'utiliser une bibliothèque de test d'interaction type mockito, ces bibliothèques ne permettant pas tout (par ex mockito ne permet pas de mocker les types natifs / built-in python).

Les entrées sorties Python Tout en haut de la hiérarchie des classes dédiées aux es (paquetage `io`) on trouve la classe abstraite `IOBase`, sous-classée par `BufferedIOBase`, puis par `TextIOWrapper` (le type de `sys.stdin` et `sys.stdout`, et aussi le type retourné quand on ouvre un fichier par `open`) et `StringIO`. `StringIO`, comme son nom l'indique, permet d'utiliser des chaînes.



Dans les tests on utilisera un objet de type `StringIO` :

— pour les entrées, le flot est passé à la construction de l'objet sous la forme d'une chaîne :

```
# pour les tests
>>> from io import StringIO
>>> e = StringIO("1\n2\n")
>>> e.readline()
'1\n'
>>> e.readline()
'2\n'
>>> e.readline()
''
```

```
# pour l'application, toto.txt
# contient 1 en 1ere ligne, 2 en 2nde ligne
>>> e = open("toto.txt")
>>> e.readline()
'1\n'
>>> e.readline()
'2\n'
>>> e.readline()
''
```

— pour les sorties, le texte écrit dans l'objet de type `StringIO` est récupéré par la méthode `getvalue()` pour utilisation dans une assertion.

```
>>> e = StringIO()
>>> e.write("foo\n")
4
>>> e.getvalue()
'foo\n'
```

Les entrées sorties Java Un aperçu du paquetage `java.io` est donné figure 1. Les super-type sont `InputStream` et `OutputStream`, qui seront les types injectés dans l'application. `System.in` est de type `InputStream` qui fournit `read()`. `System.out` est de type `PrintStream` qui fournit les méthodes en `print*`. `OutputStream` fournit `write(...)`.

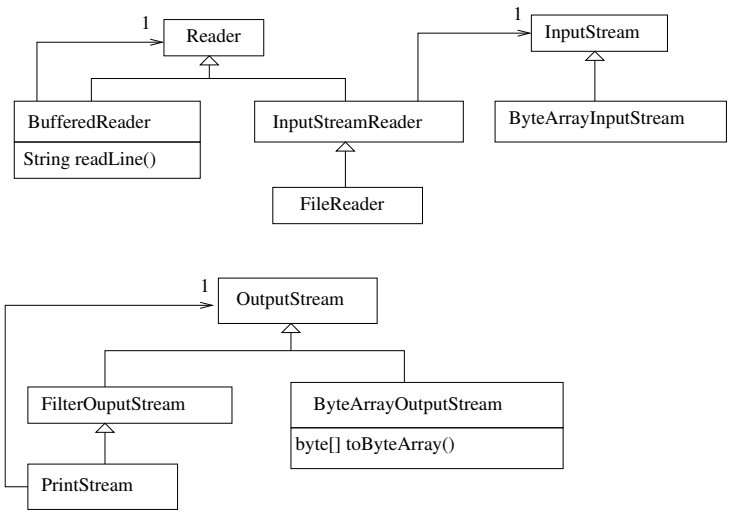


FIGURE 1 – Extrait du paquetage `java.io`

Les entrées Comment lire des entrées à partir d'une donnée de test dans un cas de test, et non à partir du clavier ou d'un fichier ? La manière traditionnelle de lire un flot d'entrée de manière efficace est d'utiliser un `Reader` tamponné de type `BufferedReader` qui fournit `readLine()`. On écrira par exemple :

```
BufferedReader reader = new BufferedReader(new InputStreamReader(System.in));
String string_line = reader.readLine();
```

Pour injecter la dépendance au flot, on paramètre l'application par un `InputStream in`. Dans l'application, on utilise un reader construit à partir de `in` (remplacer `System.in` par `in`).
Dans les tests, on peut construire utiliser la classe `ByteArrayInputStream` (Javadoc : *A ByteArrayInputStream contains an internal buffer that contains bytes that may be read from the stream*), et construire un flot d'entrée à partir d'une chaîne de caractères de la manière suivante :

```
String input_data = "ma_donnee_de_test\n";
ByteArrayInputStream input = new ByteArrayInputStream(input_data.getBytes());
```

Il est alors possible de remplacer lors des appels le flot d'entrée `in` de l'application par le flot de test `input`.

Les sorties Comment contrôler dans un cas de test les sorties que l'application émet sur la console ou dans un fichier ? Pour injecter la dépendance au flot, on paramètre l'application par un `OutputStream out`. Dans l'application, on utilise par exemple un `PrintStream` :

```
PrintStream outputStream = new PrintStream(out);
```

Dans les test, on peut utiliser la classe `ByteArrayOutputStream` (Javadoc : *This class implements an output stream in which the data is written into a byte array. [...] The data can be retrieved using toByteArray() and toString().*), et transformer en `String` le flot de sorties de la manière suivante :

```
ByteArrayOutputStream stringableOutput = new ByteArrayOutputStream();
PrintStream out_test = new PrintStream(stringableOutput);
```

Il est alors possible de remplacer lors des appels le flot de sortie `out` de l'application par le flot de test `out_test` et de récupérer ensuite ce qui a été écrit dedans par :

```
String effective_output = new String(stringableOutput.toByteArray());
```

Les oracles peuvent utiliser cette chaîne. Si on s'attend à ce que le code exécuté par le test écrive `"toto"` dans le flot `out`, on vérifiera :

```
assertEquals("toto", effective_output);
```