Formal Methods in Software Development
Exercise 5 (December 17)

Wolfgang Schreiner
Wolfgang.Schreiner@risc.uni-linz.ac.at

November 18, 2007

The result is to be submitted by the deadline stated above via the Moodle interface as a .zip or .tgz file which contains

- A PDF file with
  - a cover page with the title of the course, your name, Matrikelnummer, and email-address,
  - the JML-annotated Java code, and a copy of the output of the `escjava2` check of that code,
  - optionally any explanations or comments you would like to make;
- the JML-annotated Java (.java) files developed in the exercise.

5(all): JML Specification of a Dictionary Class

A *dictionary* is an object that maps keys to values. In Java, the abstract class `Dictionary` in package `java.util` provides a sample interface for dictionary classes (see the appendix).

First, create an abstract class `Dictionary1` with the same interface as the class `Dictionary` but let the methods `elements()` and `keys()` return `Object[]` rather than `Enumeration`. Then derive from the abstract class `Dictionary1` a concrete class `ArrayDictionary1` that implements the dictionary by two private arrays for the keys and the elements and provide an adequate private specification for this class.

Second, change `Dictionary1` to an abstract class `Dictionary2` with an adequate public JML contract. This contract shall make use of of an axiomatically specified model class `DictionaryModel` (see below). Provide an adequate model implementation and specification of the abstraction function `toModel`.

Third, change `ArrayDictionary1` to a new class `ArrayDictionary2` that inherits from `Dictionary2` such that it (in addition to its private specification) implements the public contract.

Type-check all JML-annotated files with `jml` and check them with `escjava2` (you may switch off any warnings about unsatisfied postconditions in classes `DictionaryModel`, `Dictionary2` and `ArrayDictionary2`).

The result of the exercise shall include the source codes of both versions of `Dictionary` and `ArrayDictionary` and of `DictionaryModel` together with the outputs of the `escjava2` checks of these files.

Hint: an abstract datatype D of dictionaries with keys of type K and elements of type E has the operations

$$
\begin{aligned}
&\mathit{empty} : D \\
&\mathit{isEmpty} : D \to \mathbb{B} \\
&\mathit{size} : D \to \mathbb{N} \\
&\mathit{get} : D \times K \to E \\
&\mathit{put} : D \times K \times E \to D \\
&\mathit{keys} : D \to K^* \\
&\mathit{elements} : K \to E^*
\end{aligned}
$$

Equip `DictionaryModel` with corresponding methods including their adequate axiomatization (see the example `StackModel` presented in class). You may omit the axiomatization of *keys* and *elements*.

java.util

Class Dictionary

```
java.lang.Object
  └ java.util.Dictionary
```

Direct Known Subclasses:

Hashtable

---

public abstract class **Dictionary**
extends Object

The `Dictionary` class is the abstract parent of any class, such as `Hashtable`, which maps keys to values. Every key and every value is an object. In any one `Dictionary` object, every key is associated with at most one value. Given a `Dictionary` and a key, the associated element can be looked up. Any non-`null` object can be used as a key and as a value.

As a rule, the `equals` method should be used by implementations of this class to decide if two keys are the same.

NOTE: This class is obsolete. New implementations should implement the Map interface, rather than extending this class.

Since:

JDK1.0

See Also:

`Map`, `Object.equals(java.lang.Object)`, `Object.hashCode()`, `Hashtable`

Constructor Summary

`Dictionary``()`	Sole constructor.

Method Summary

abstract Enumeration	**`elements`**`()` Returns an enumeration of the values in this dictionary.
abstract Object	**`get`**`(Object key)` Returns the value to which the key is mapped in this dictionary.
abstract boolean	**`isEmpty`**`()` Tests if this dictionary maps no keys to value.

abstract Enumeration	**keys**() Returns an enumeration of the keys in this dictionary.
abstract Object	**put**(Object key, Object value) Maps the specified `key` to the specified `value` in this dictionary.
abstract Object	**remove**(Object key) Removes the `key` (and its corresponding `value`) from this dictionary.
abstract int	**size**() Returns the number of entries (dinstint keys) in this dictionary.

Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

Dictionary

```
public Dictionary()
```

Sole constructor. (For invocation by subclass constructors, typically implicit.)

Method Detail

size

```
public abstract int size()
```

Returns the number of entries (dinstint keys) in this dictionary.

Returns:
the number of keys in this dictionary.

---

isEmpty

```
public abstract boolean isEmpty()
```

Tests if this dictionary maps no keys to value. The general contract for the `isEmpty` method is that the result is true if and only if this dictionary contains no entries.

Returns:
`true` if this dictionary maps no keys to values; `false` otherwise.

---

keys

```
public abstract Enumeration keys()
```

Returns an enumeration of the keys in this dictionary. The general contract for the keys method is that an `Enumeration` object is returned that will generate all the keys for which this dictionary contains entries.

Returns:
an enumeration of the keys in this dictionary.

See Also:
`elements()`, `Enumeration`

---

elements

```
public abstract Enumeration elements()
```

Returns an enumeration of the values in this dictionary. The general contract for the `elements` method is that an `Enumeration` is returned that will generate all the elements contained in entries in this dictionary.

Returns:
an enumeration of the values in this dictionary.

See Also:
`keys()`, `Enumeration`

---

get

```
public abstract Object get(Object key)
```

Returns the value to which the key is mapped in this dictionary. The general contract for the `isEmpty` method is that if this dictionary contains an entry for the specified key, the associated value is returned; otherwise, `null` is returned.

Parameters:
`key` - a key in this dictionary. `null` if the key is not mapped to any value in this dictionary.

Returns:
the value to which the key is mapped in this dictionary;

Throws:
`NullPointerException` - if the `key` is `null`.

See Also:
`put(java.lang.Object, java.lang.Object)`

---

put

```
public abstract Object put(Object key,
                           Object value)
```

Maps the specified `key` to the specified `value` in this dictionary. Neither the key nor the value can be `null`.

If this dictionary already contains an entry for the specified `key`, the value already in this dictionary for that `key` is returned, after modifying the entry to contain the new element.

If this dictionary does not already have an entry for the specified `key`, an entry is created for the specified `key` and `value`, and `null` is returned.

The `value` can be retrieved by calling the `get` method with a `key` that is equal to the original `key`.

Parameters:

`key` - the hashtable key.
`value` - the value.

Returns:

the previous value to which the `key` was mapped in this dictionary, or `null` if the key did not have a previous mapping.

Throws:

`NullPointerException` - if the `key` or `value` is `null`.

See Also:

`Object.equals(java.lang.Object)`, `get(java.lang.Object)`

---

remove

```
public abstract Object remove(Object key)
```

Removes the `key` (and its corresponding `value`) from this dictionary. This method does nothing if the `key` is not in this dictionary.

Parameters:

`key` - the key that needs to be removed.

Returns:

the value to which the `key` had been mapped in this dictionary, or `null` if the key did not have a mapping.

Throws:

`NullPointerException` - if `key` is `null`.
