17 MAPS

Terry Marris 28 May 2001

17.1 OBJECTIVES

By the end of this lesson the student should be able to

- discuss the notion of a map
- use the HashMap methods

17.2 PRE-REQUSITES

The student should be comfortable with using sets (Chapter 12) and with implementing associations (Chapter 13).

17.3 PREVIEW

We look at the concept of a map. We examine the *HashMap* class. We see how a *HashMap* implements a map.

17.4 INTRODUCTION

A set is a collection that lets you find an existing element, but to look up that element you need to have an exact copy of it; hardly convenient.

Usually, you would know a key and want to find the corresponding element. For example, knowing a student reference number you want to retrieve that student's record.

The map data structure allows you to do just that.

17.5 KEY-VALUE PAIR

We are all familiar with the notion that no two students have the same reference number, no two vehicles have the same registration mark and a consultant cannot (should not) have two different appointments at the same time.

Student reference number, registration mark and appointment date and time are all examples of keys because they uniquely identify a value.

A student value might be *number = 007, name = bond, gender = male, subject = computing.*

A vehicle value might be *index mark = X 678 GDS, make = Audi, colour = silver*.

An appointment value might be *date = 29 May 2001, time = 90:30, with = Mr Bond*.

A key uniquely identifies a value.

17.6 MAP

A Java map stores key-value pairs where

each key has just one value associated with it and

no two keys are the same

For example

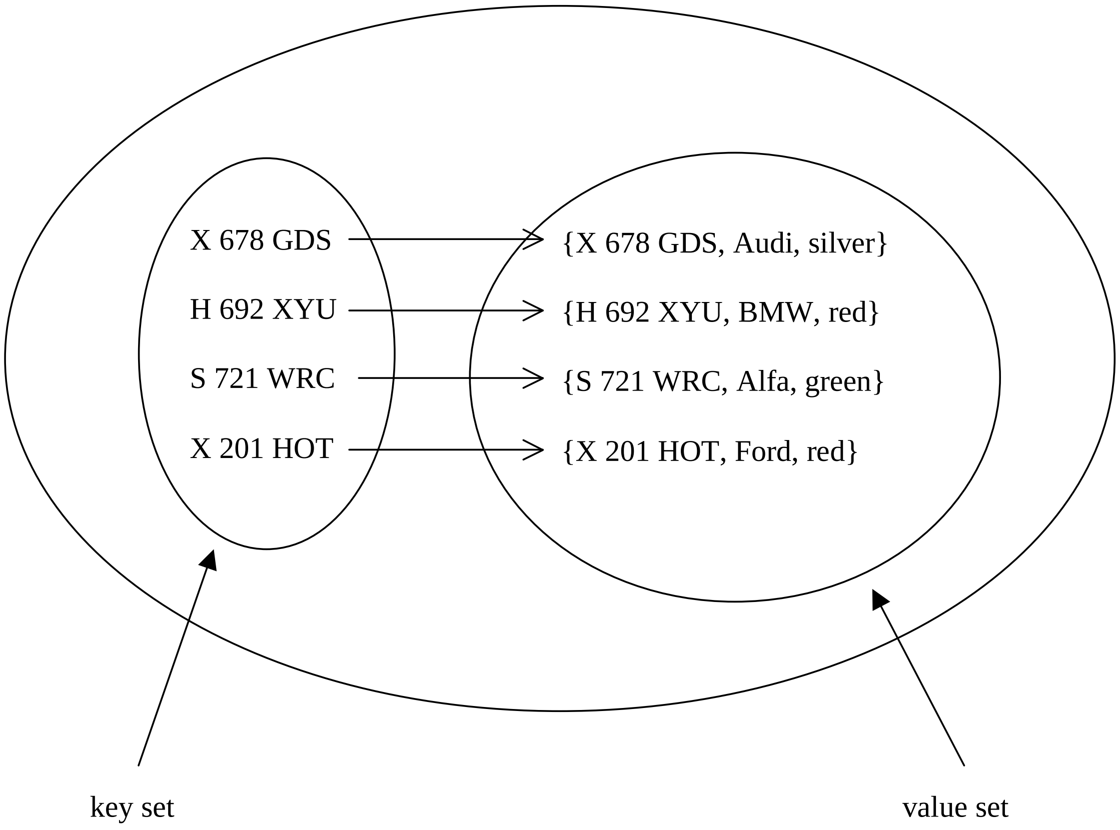


Figure 17.1 *A Map*

A key must map to just one value. But a value may have more than one key mapped to it. For example, in Figure 19.2, H692XYU and E823DRY both map to a red BMW.

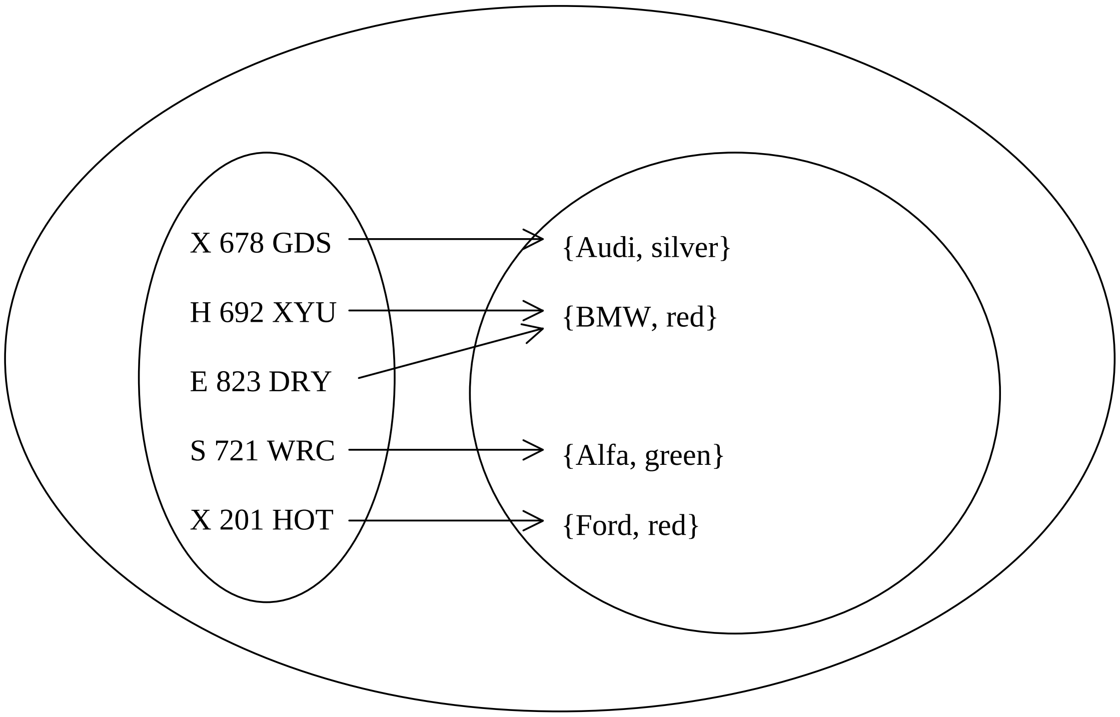


Figure 17.2 *Keys must be unique and map to just one value. A value may have more than one key mapped to it.*

17.7 USING A HASH MAP

We shall use a hash map to organise a collection of students. Each student will be uniquely identified by his or her number.

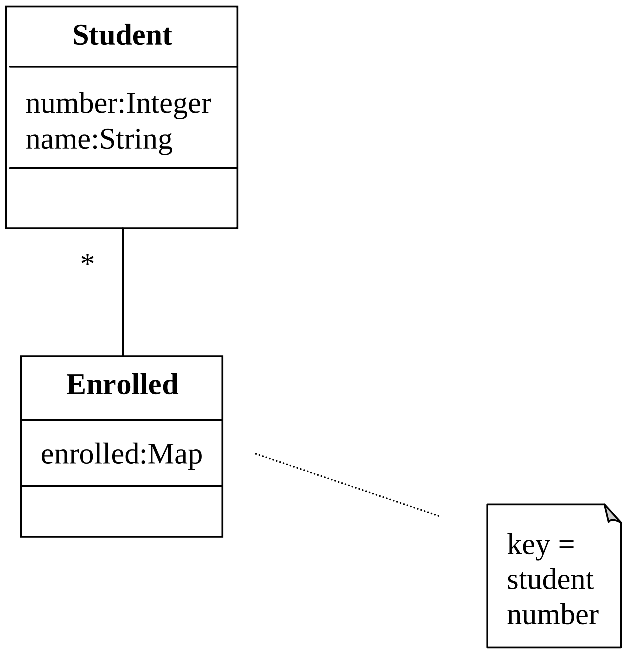


Figure 17.3 *Enrolled maintains a map of studentNumber-Student pairs*

Student is our regular *Student* class. When a new *Student* object is created, the next new number is also generated and assigned to the number field. This number will serve as our key. Since sets cannot contain primitive types such as *int*s, we define the number field as an *Integer*. The Student class has methods to retrieve the student number and to set the student's name.

```
class Student {
  private static int nextNumber = 1;
  private Integer number;
  private String name;

  public Student(String aName)
  {
    number = new Integer(nextNumber);
    nextNumber++;
    name = aName;
  }


  public Integer getNumber()
  {
    return number;
  }


  public void setName(String aName)
  {
    name = aName;
  }


  public String toString()
  {
    return "Number: " + number + ", Name: " + name;
  }
}
```

The *Enrolled* class diagram specified that the attribute *enrolled* is a *Map*. We shall implement *enrolled* as a *HashMap*.

Declaring and initialising *enrolled* is straightforward. Initially, *enrolled* has no entries.

```
private Map enrolled;


public Enrolled()
{
  enrolled = new HashMap();
}
```

The *add()* method creates a new *Student* object (with its own unique self-generated student number) from the given name before putting it into the map.

```
 public String add(String name)
{
  Student student = new Student(name);
```

We check that our map does not already have a key-value entry with the same student number: this should not happen.

```
if (enrolled.containsKey(student.getNumber()))
  return "failure - student already enrolled";
```

Whenever you add an object to a map you must supply a key as well. We use the *put()* method to place a key-value pair (number-student) into the map.

```
enrolled.put(student.getNumber(), student);
```

To retrieve an object from the map, you must use the key. The key is supplied as the parameter to the *get()* method.

```
if (enrolled.containsKey(studentNumber))
  return (Student)enrolled.get(studentNumber);
```

The *remove()* method removes the key-value pair for the given student number. You can only remove a pair if the key value exists.

```
if (enrolled.containsKey(studentNumber)) {
  enrolled.remove(studentNumber);
```

The *put()* method, as well as adding a new pair to the map, also replaces the pair with the given key. So we first check that a pair with the given key exists.

```
if (enrolled.containsKey(student.getNumber())) {
  enrolled.put(student.getNumber(), student);
```

To iterate over the entire map we first have to obtain the key-value entries as a set.

```
Set entries = enrolled.entrySet();
```

Then we set up an iterator over the set.

```
Iterator it = entries.iterator();
while (it.hasNext()) {
```

An *entrySet* has *Map.Entry* objects as its elements. To retrieve the next map entry we use

```
Map.Entry entry = (Map.Entry)it.next();
```

Then we extract the value part of a map entry pair with

```
Object value = entry.getValue();
```

Having got the value part as an object, we use the cast operator to obtain a student.

```
Student student = (Student)value;
```

Now we can do what we like with the student.

Here is the *Enrolled* class in more detail

```
public class Enrolled {
  private Map enrolled;


  public Enrolled()
  {
    enrolled = new HashMap();
  }


  public String add(String name)
  {
    Student student = new Student(name);
    if (enrolled.containsKey(student.getNumber()))
      return "failure - student already enrolled";
    enrolled.put(student.getNumber(), student);
    return "success";
  }


  public Student find(Integer studentNumber)
  {
    if (enrolled.containsKey(studentNumber))
      return (Student)enrolled.get(studentNumber);
    return null;
  }


  public String remove(Integer studentNumber)
  {
    if (enrolled.containsKey(studentNumber)) {
      enrolled.remove(studentNumber);
      return "success";
    }
    return
          "failure - student with the given number not found";
  }


  public String replace(Student student)
  {
    if (enrolled.containsKey(student.getNumber())) {
      enrolled.put(student.getNumber(), student);
      return "success";
    }
    return "failure - given student not found";
  }
```

```
public String toString()
{
  String string = "";
  Set entries = enrolled.entrySet();
  Iterator it = entries.iterator();
  while (it.hasNext()) {
    Map.Entry entry = (Map.Entry)it.next();
    Object value = entry.getValue();
    Student student = (Student)value;
    string += student;
    if (it.hasNext())
      string += "\n";
  }
  return string;
}
```

Exercise: Explain what each method of the *Enrolled* class shown above does. Explain how each method accomplishes its task.

We test out some of the *Enrolled* class methods.

We create an instance of the *Enrolled* class, add five new students to it, and then print out the contents of the map.

```
    Enrolled studentBody = new Enrolled();
    studentBody.add("tom");
    studentBody.add("anne");
    studentBody.add("jerry");
    studentBody.add("homer");
    studentBody.add("madge");

    System.out.println("After adding five students ... ");
    System.out.println(studentBody);
```

The output on the screen is

```
    After adding five students ...
    Number: 5, Name: madge
    Number: 4, Name: homer
    Number: 3, Name: jerry
    Number: 2, Name: anne
    Number: 1, Name: tom
```

Like a set, a *HashMap* does not guarantee the order in which entries are stored and retrieved.

Then we remove students with keys one and five.

```
System.out.println("After removing student #1 and #5 ... ");
studentBody.remove(new Integer(1));
studentBody.remove(new Integer(5));
System.out.println(studentBody);
```

The output is

```
    After removing student #1 and #5 ...
    Number: 4, Name: homer
    Number: 3, Name: jerry
    Number: 2, Name: anne
```

Then we change the student number 3's name from jerry to terry.

```
System.out.println("After finding student #3 and " +
            "and changing its name from jerry to terry ...");
Student student = studentBody.find(new Integer(3));
student.setName("terry");
studentBody.replace(student);
System.out.println(studentBody);
```

The output is

```
     Number: 4, Name: homer
     Number: 3, Name: terry
     Number: 2, Name: anne
```

The entire program is shown below.

```
/* Enrolled.java
   Terry Marris  29 May 2001
*/

import java.util.*;

class Student {
  private static int nextNumber = 1;
  private Integer number;
  private String name;

  public Student(String aName)
  {
    number = new Integer(nextNumber);
    nextNumber++;
    name = aName;
  }


  public Integer getNumber()
  {
    return number;
  }


  public void setName(String aName)
  {
    name = aName;
  }


  public String toString()
  {
    return "Number: " + number + ", Name: " + name;
  }
}
```

```
public class Enrolled {
  private Map enrolled;

  public Enrolled()
  {
    enrolled = new HashMap();
  }


  public String add(String name)
  {
    Student student = new Student(name);
    if (enrolled.containsKey(student.getNumber()))
      return "failure - student already enrolled";
    enrolled.put(student.getNumber(), student);
    return "success";
  }


  public Student find(Integer studentNumber)
  {
    if (enrolled.containsKey(studentNumber))
      return (Student)enrolled.get(studentNumber);
    return null;
  }


  public String remove(Integer studentNumber)
  {
    if (enrolled.containsKey(studentNumber)) {
      enrolled.remove(studentNumber);
      return "success";
    }
    return "failure - student with the given number not found";
  }


  public String replace(Student student)
  {
    if (enrolled.containsKey(student.getNumber())) {
      enrolled.put(student.getNumber(), student);
      return "success";
    }
    return "failure - given student not found";
  }

  public String toString()
  {
    String string = "";
    Set entries = enrolled.entrySet();
    Iterator it = entries.iterator();
    while (it.hasNext()) {
      Map.Entry entry = (Map.Entry)it.next();
      Object value = entry.getValue();
      Student student = (Student)value;
      string += student;
      if (it.hasNext())
        string += "\n";
    }
    return string;
  }
  public static void main(String[] s)
  {
    Enrolled studentBody = new Enrolled();
    studentBody.add("tom");
    studentBody.add("anne");
    studentBody.add("jerry");
    studentBody.add("homer");
    studentBody.add("madge");

    System.out.println("After adding five students ... ");
    System.out.println(studentBody);

    System.out.println("After removing student #1 and #5 ... ");
    studentBody.remove(new Integer(1));
    studentBody.remove(new Integer(5));
    System.out.println(studentBody);

    System.out.println("After finding student #3 and " +
               "and changing its name from jerry to terry ...");
    Student student = studentBody.find(new Integer(3));
    student.setName("terry");
    studentBody.replace(student);
    System.out.println(studentBody);
  }
}
```

Output:

```
After adding five students ...
Number: 5, Name: madge
Number: 4, Name: homer
Number: 3, Name: jerry
Number: 2, Name: anne
Number: 1, Name: tom
After removing student #1 and #5 ...
Number: 4, Name: homer
Number: 3, Name: jerry
Number: 2, Name: anne
After finding student #3 and changing its name from jerry to terry ...
Number: 4, Name: homer
Number: 3, Name: terry
Number: 2, Name: anne
```

17.8 THE HASH MAP CLASS

HashMap implements *Map*.

java.util.HashMap

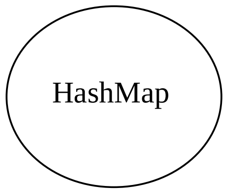
Constructors	
HashMap()	Initialises a new, empty HashMap.
HashMap(Map t)	Initialises a new HashMap with the given map.

Methods		
void	clear()	removes all mappings from this map.
Object	clone()	returns a copy of this HashMap instance, the keys and values themselves are not cloned.
boolean	containsKey(Object key)	returns true of this HashMap contains the given key.
boolean	containsValue(Object value)	returns true if this map maps one or more keys to the given value.
Set	entrySet()	returns a collection view of the mappings contained in this map.
Object	get(Object key)	returns the value to which this map maps the given key.
boolean	isEmpty()	returns true if this map contains no key-value mappings.
Set	keySet()	returns a set view of the keys contained in this map.
Object	put(Object key, Object value)	associates the given value with the given key in this map.
void	putAll(Map t)	copies all the mappings from the given map to this one.
Object	remove(Object key)	removes the mapping for the given key from this map, if present.
int	size()	returns the number of key-value mappings in this map.
Collection	values()	returns a collection view of the values contained in this map.

17.9 FURTHER READING

HORSTAMNN & CORNELL *Core Java 2 Volume 2* pp 105
www.java.sun.com/docs/books/tutorial/collections

17.10 REVIEW



use when you want to retrieve data on a key value

stores key-value pairs

put() either adds or replaces the given key-value pair

get() retrieves the value for the given key

contains() returns true if an entry with the given key exists

remove() removes the entry with the given key

entrySet() returns a set view of the entries

Map.Entry is an element of an *entrySet*

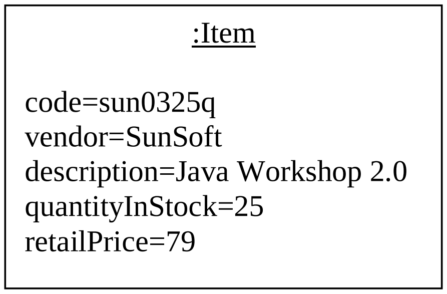
getValue() returns the value part of an *entrySet* element.

17.11 EXERCISES

1 Explain the key features of a map. In what circumstances would you consider using a map?

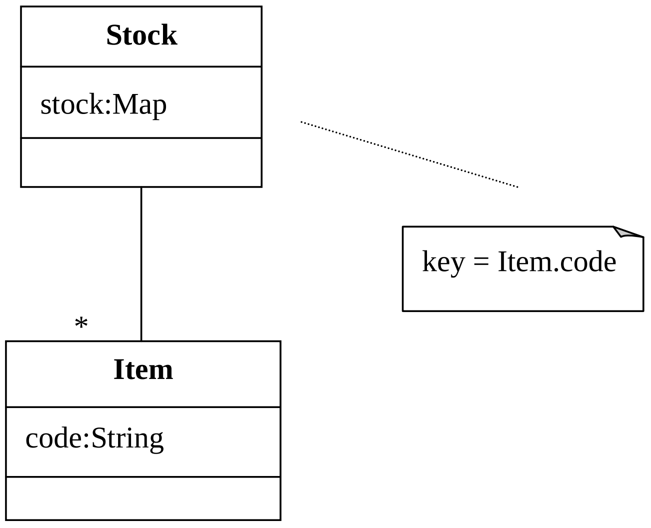
2 List and explain the *HashMap* methods that allow you to add, find, remove, update and list key-value pairs.

3 A specialist retailer stocks software. An example of a stock *Item* instance is



An *Item* has a unique code, a description, a vendor (e.g. MicroSoft, Enprise, Borland, Sun), a description, a quantity in stock and a retail price.

Stock has a map of Item objects.



(a) Implement and test *Item* with the usual constructors, setting, getting and *toString()* methods.

(b) Implement and test each of the following *Stock* methods.

Stock

Fields	
Map stock	stock is a HashMap instance.

Constructors	
Stock()	initialises stock to empty.

Methods		
String	add(String code, String vendor, String description, int initialQuantity)	creates a new stock Item with the given code, vendor, description and initial quantity. The initial quantity cannot be negative. The code cannot already exist in stock. The description cannot already exist in stock. Returns success if all conditions are met and the new stock Item is added to stock, returns failure otherwise.
String	decreaseQuantity(String code, int amount)	decreases the quantity in stock for the item with the given code by the given amount. Returns success if the item with the given code is successfully updated, returns failure if the given code cannot be found in stock.
String	priceList()	returns a string that lists each item in stock, its code, vendor, description and retail price.