

Arrays and Maps

Arrays

An array is a collection of objects of the same type. It is:

- ordered
- indexed
- fixed length

Arrays and ArrayLists

Arrays:

- Are **fixed** size
- Are **built-in** to Java
- Can contain **primitive** types or **objects**

ArrayLists:

- Are **variable** size
- Are part of the **Java class library**
- Can only contain **objects**

The ArrayList class uses arrays internally.

Array definitions

We declare a variable that holds an array as:

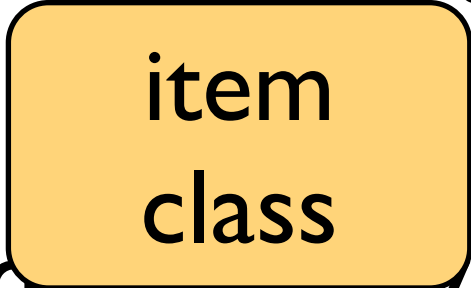
```
String[] arrayOfStrings;
```

```
int[] arrayOfInts;
```

```
Book[] arrayOfBooks;
```

Array definitions

item
class



We declare a variable that holds an array as:

```
String[] arrayOfStrings;
```

```
int[] arrayOfInts;
```

```
Book[] arrayOfBooks;
```

Array definitions

item
class

We declare a variable that holds an array as:

```
String[] arrayOfStrings;
```

```
int[] arrayOfInts;
```

```
Book[] arrayOfBooks;
```

square
brackets

Construction

To construct an array we need to specify how many elements it contains:

```
// create an array of 3 items
```

```
String[] classRoll =  
    new String[3];
```

0	null
1	null
2	null

Three entries are created and initialised to null (or zero if they are numbers).

Construction

The number can be the result of an expression computed at run-time:

```
int nStudents =  
    myUNSW.getClassSize("COMP1400");  
String[] classRoll =  
    new String[nStudents];
```

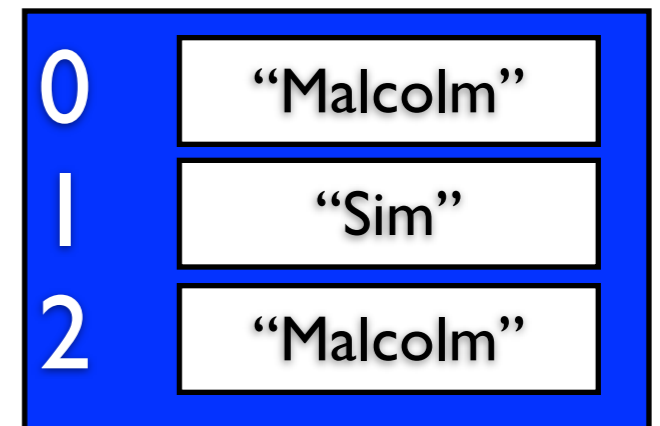
Reading and writing

Each entry in the array can be considered as a separate variable, accessed by its index:

```
classRoll[0] = "Malcolm";
```

```
classRoll[1] = "Sim";
```

```
classRoll[2] =  
    classRoll[0];
```



Note that the index starts at **zero**.

Array length

We can find out the length of an array by accessing the **length** field.

```
int nStudents =  
    classRoll.length;  
  
// nStudents == 3
```

Note that this field is **read-only**.

```
classRoll.length = 4; // ERROR
```

0	"Malcolm"
1	"Malcolm"
2	null

For loops and arrays

The `for-each` loop does **not** work on arrays.

```
for (String who : classRoll) {  
    // ERROR: will not compile  
}
```

For loops and arrays

Instead, we commonly use the **for** loop to iterate over an array:

```
for (int i = 0;  
      i < classRoll.length;  
      i++) {  
    // Perform an operation  
    // on classRoll[i]  
}
```

HashMaps

A HashMap is a collection that associates two kinds of objects, a **key** and a **value**.

Examples are:

- A dictionary
(**key** = word, **value** = meaning)
- A phone book
(**key** = name, **value** = number)

HashMaps

A HashMap is:

- unordered
- index by keys
- variable size

Note: there can be only one value for each key, but values may be repeated for different keys.

HashMap

The HashMap is part of the Java class library and so must be imported before it is used:

```
import java.util.HashMap;
```

The API docs can be found at:

<http://download.oracle.com/javase/6/docs/api/java/util/HashMap.html>

HashMap definitions


HashMaps take **two** type parameters -- the **key type** and the **value type**:

```
HashMap<String, Integer>  
    mapStringToNumber;
```

```
HashMap<Book, Author>  
    mapBookToAuthor;
```

As with ArrayLists, we cannot have HashMaps of primitive types (either key or value).

HashMap definitions

HashMap  **two** type parameters --
the **key type** and the **value type**:

```
HashMap<String, Integer>  
    mapStringToNumber;
```

```
HashMap<Book, Author>  
    mapBookToAuthor;
```

As with ArrayLists, we cannot have HashMaps of primitive types (either key or value).

HashMap definitions

HashMap parameters --
the **key type** and the **value type**:

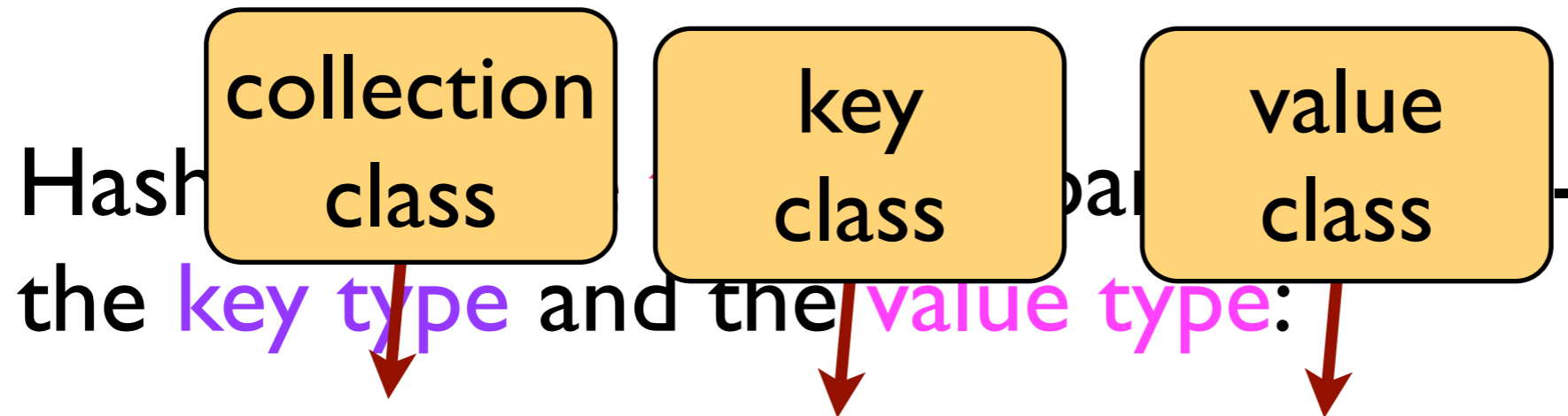
The diagram consists of two yellow rounded rectangular boxes. The left box contains the text 'collection class' and the right box contains 'key class'. Below the left box, a red arrow points down to the text 'key type' in purple. Below the right box, a red arrow points down to the text 'value type' in pink.

```
HashMap<String, Integer>  
mapStringToNumber;
```

```
HashMap<Book, Author>  
mapBookToAuthor;
```

As with ArrayLists, we cannot have HashMaps of primitive types (either key or value).

HashMap definitions



```
HashMap<String, Integer>  
mapStringToNumber;
```

```
HashMap<Book, Author>  
mapBookToAuthor;
```

As with ArrayLists, we cannot have HashMaps of primitive types (either key or value).

Construction

To construct an `HashMap` we will usually use the simple no-parameter constructor:

```
// create an empty map
```

```
HashMap<String, Int> marks =  
    new HashMap<String, Integer>();
```



key (String) value (Integer)

Adding items

A newly constructed HashMap is **empty**.

We can add items using the **put** method.

```
marks.put("Malcolm", 90);  
marks.put("Claude", 55);
```

key (String)	value (Integer)
"Malcolm"	90
"Claude"	55

Overwriting items

Each key has only **one** value.

Adding a new one overwrites the old:

```
marks.put("Claude", 65);
```

key (String)	value (Integer)
"Malcolm"	90
"Claude"	65

Getting items

We can read items using the `get` method.

```
marks.get("Malcolm");
```

```
// returns 90
```

```
marks.get("Sim");
```

```
// returns null
```

key (String)	value (Integer)
"Malcolm"	90
"Claude"	55

Integers, ints and null

An important difference between Integers and `ints`.

Integer variables are object references and so they can be `null`.

`int` variables are numbers and `cannot` be `null`.

Integers, ints and null

This sometimes causes problems with maps:

```
int mark =  
    marks.get("Malcolm");  
  
// Integer 90 is converted to int  
  
mark =  
    marks.get("Sim");  
  
// ERROR:  
// Cannot convert null to int
```

Solution

If there is doubt, always check the value first:

```
Integer mark =  
    marks.get("Sim");  
  
if (mark == null) {  
    // handle the null case  
}  
else {  
    // handle non-null case  
}
```

Iterating

We **cannot** use the for-each loop directly on the HashMap:

```
for (String who : marks) {  
    // ERROR: will not compile  
}
```

Iterating

However we **can** iterate over the **key set**:

```
for (String who : marks.keySet())  
{  
    Integer value = marks.get(who);  
    // do something with who/mark  
}
```

Other methods

There are other methods that you should read about in the API:

- `containsKey` - check if a key is in the map
- `size` - how many entries in the map
- `remove` - remove an entry by key

Practise reading the API docs.