

Java^{SE 7}

技術手冊

林良

- 涵蓋 OCP/JP (原 SCJP) 考試範圍
- Coin 專案、JSR166y、JDBC 4.1、NIO.2 等 Java SE 7 新功能介紹
- JDK 基礎與 IDE 操作交相對照
- 提供 Lab 檔案與操作錄影教學

CHAPTER 4

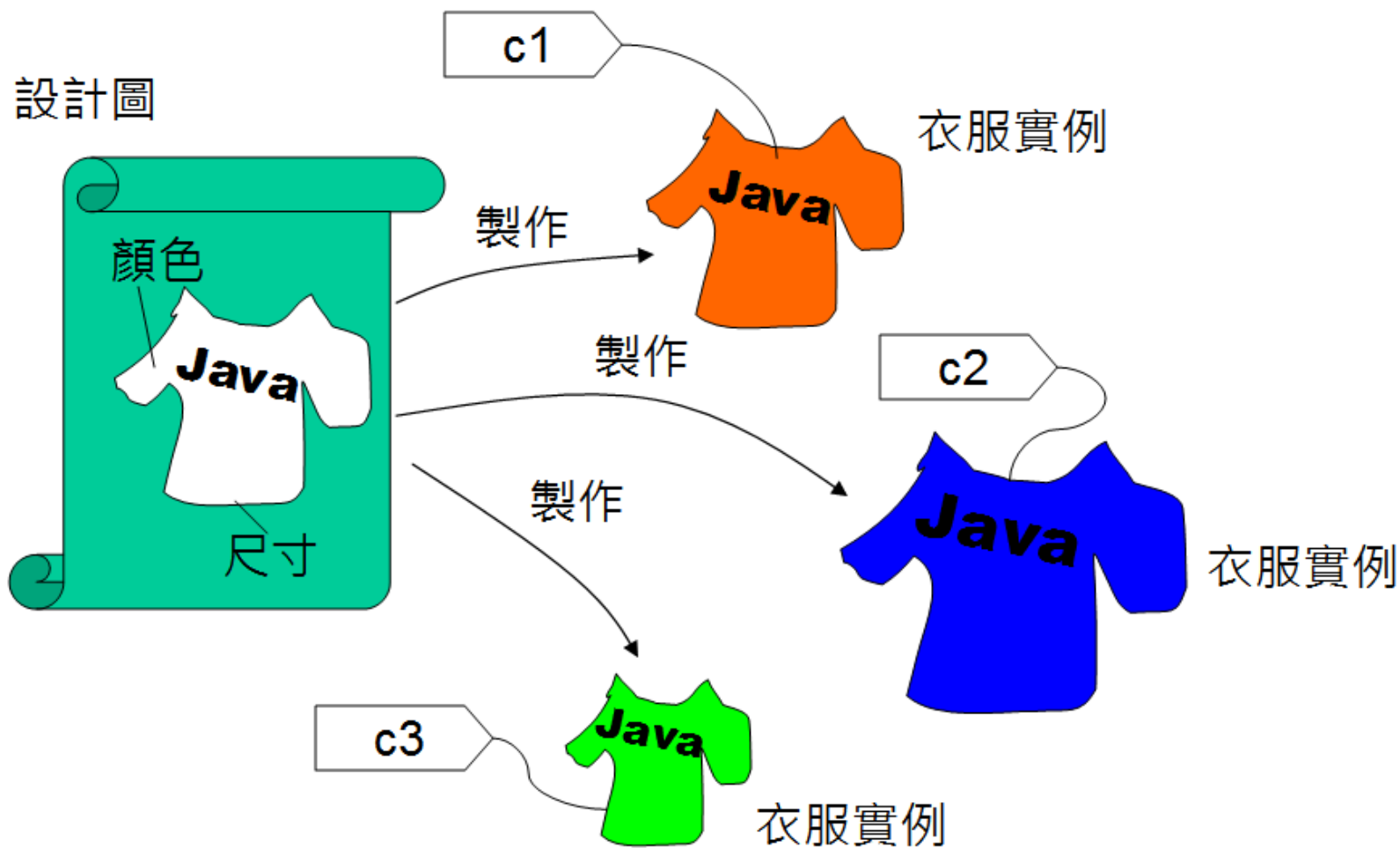
• 認識物件



學習目標

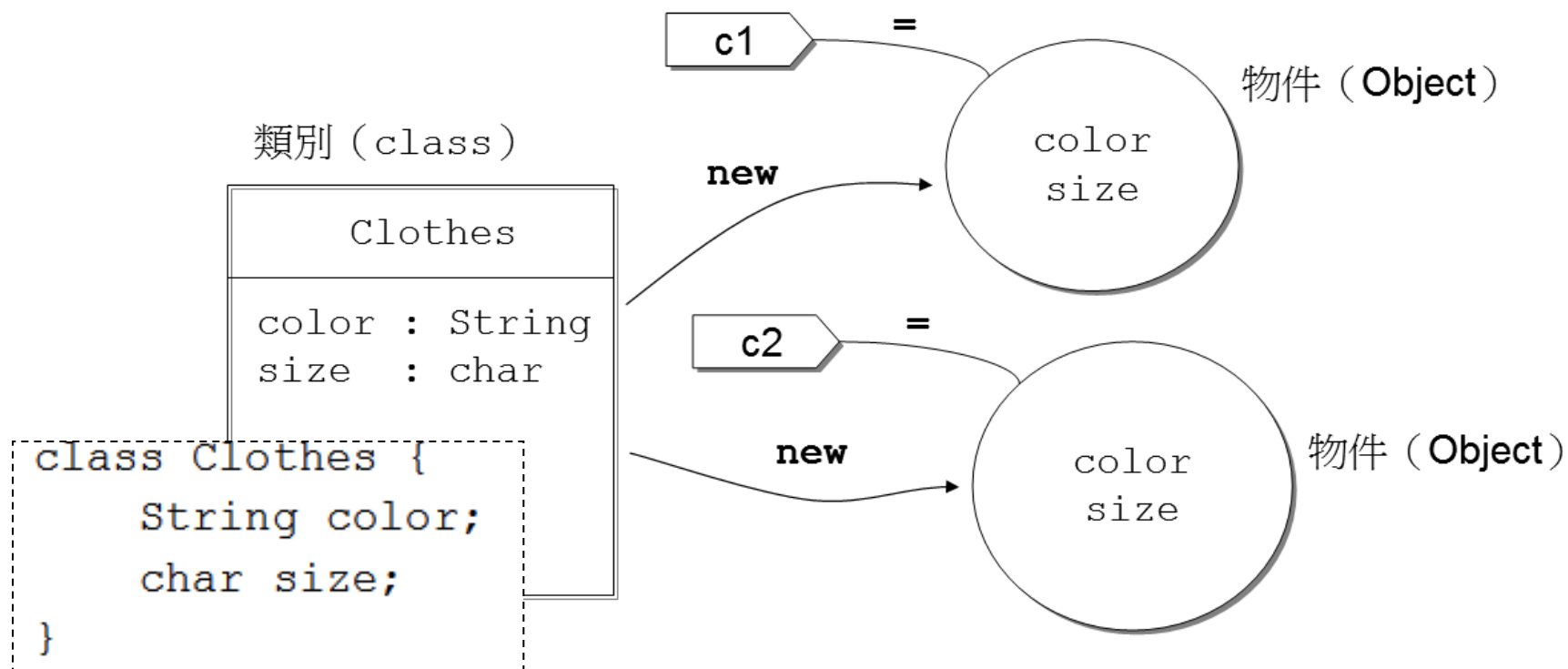
- 區分基本型態與類別型態
- 瞭解物件與參考的關係
- 從包裹器認識物件
- 以物件觀點看待陣列
- 認識字串的特性

定義類別



定義類別

```
Clothes c1 = new Clothes();
```



```
Clothes c2 = new Clothes();
```

定義類別

```
class Clothes2 {
    String color;
    char size;
    Clothes2(String color, char size) { ← ❶ 定義建構式
        this.color = color; ← ❷ color 參數的值指定給這個物件的 color 成員
        this.size = size;
    }
}

public class Field2 {
    public static void main(String[] args) {
        Clothes2 c1 = new Clothes2("red", 'S'); ← ❸ 使用指定建構式建立物件
        Clothes2 c2 = new Clothes2("green", 'M');

        System.out.printf("c1 (%s, %c)%n", c1.color, c1.size);
        System.out.printf("c2 (%s, %c)%n", c2.color, c2.size);
    }
}
```

定義類別

- 可以觀察這個範例中，為個別物件指定資料成員值的程式碼**❸**，你會發現是類似的 ..
- 如果想在建立物件時，一併進行某個初始流程，像是指定資料成員值，則可以定義建構式（ Constructor ），建構式是與類別名稱同名的方法（ Method ）

定義類別

```
import java.util.Scanner; ← ❶ 告訴編譯器接下來想偷懶

public class Guess {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in); ← ❷ 建立 Scanner 實例
        int number = (int) (Math.random() * 10);
        int guess;

        do {
            System.out.print("猜數字 (0 ~ 9) :");
            guess = scanner.nextInt(); ← ❸ 取得下一個整數
        } while(guess != number);

        System.out.println("猜中了...XD");
    }
}
```


使用標準類別

- 使用 `java.util.Scanner`

```
import java.util.Scanner; ← ❶ 告訴編譯器接下來想偷懶

public class Guess {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in); ← ❷ 建立 Scanner 實例
        int number = (int) (Math.random() * 10);
        int guess;

        do {
            System.out.print("猜數字 (0 ~ 9) :");
            guess = scanner.nextInt(); ← ❸ 取得下一個整數
        } while(guess != number);

        System.out.println("猜中了...XD");
    }
}
```


使用標準類別

- $1.0 - 0.8$ 的結果是？答案不是0.2，而是0.199999999999999996！
- Java（包括其它程式語言）符合IEEE 754浮點數演算（Floating-point arithmetic）規範

```
double a = 0.1;
double b = 0.1;
double c = 0.1;
if((a + b + c) == 0.3) {
    System.out.println("等於 0.3");
}
else {
    System.out.println("不等於 0.3");
}
```

使用標準類別

- 使用 `java.math.BigDecimal`

```
import java.math.BigDecimal;

public class DecimalDemo {
    public static void main(String[] args) {
        BigDecimal a = new BigDecimal("1.0");
        BigDecimal b = new BigDecimal("0.8");
        BigDecimal c = a.subtract(b);
        System.out.println(c);
    }
}
```

使用標準類別

- 呼叫**`equals()`** 比較兩個`BigDecimal` 實質上是否相同

```
import java.math.BigDecimal;

public class DecimalDemo2 {
    public static void main(String[] args) {
        BigDecimal a = new BigDecimal("0.1");
        BigDecimal b = new BigDecimal("0.1");
        BigDecimal c = new BigDecimal("0.1");
        BigDecimal result = new BigDecimal("0.3");
        if(a.add(b).add(c).equals(result)) {
            System.out.println("等於 0.3");
        }
        else {
            System.out.println("不等於 0.3");
        }
    }
}
```

物件指定與相等性

- 在Java中有兩大型態系統，基本型態與類別型態，這很令人困擾...
- 若不討論底層記憶體實際運作，初學者就必須區分=與==運算用於基本型態與類別型態的不同

物件指定與相等性

- 當=用於基本型態時，是將值複製給變數
- ==用於基本型態時，是比較兩個變數儲存的值是否相同

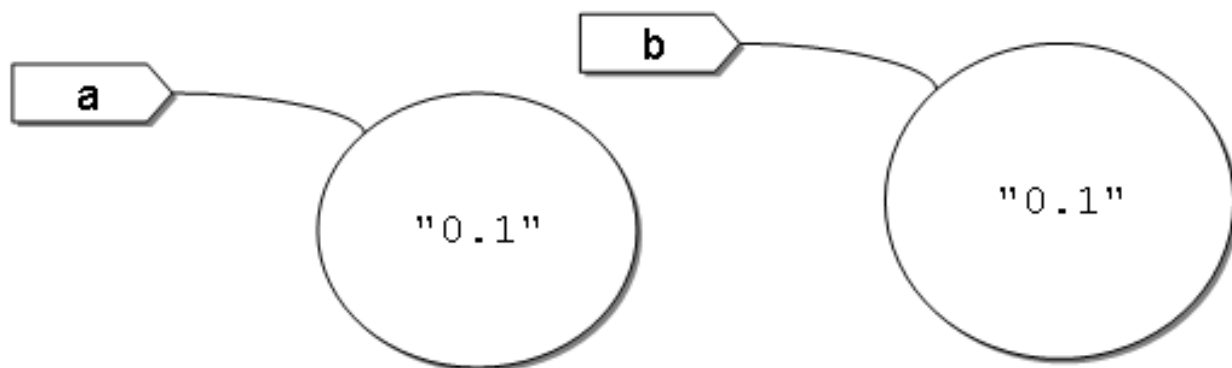
```
int a = 10;  
int b = 10;  
int c = a;  
System.out.println(a == b);  
System.out.println(a == c);
```

物件指定與相等性

- 如果你在操作物件，`=`是用在指定參考名稱參考某個物件
- `==`是比較兩個參考名稱是否參考同一物件
- 白話來說，`=`是用在將某個名牌綁到某個物件，而`==`是用在比較兩個名牌是否綁到同一物件

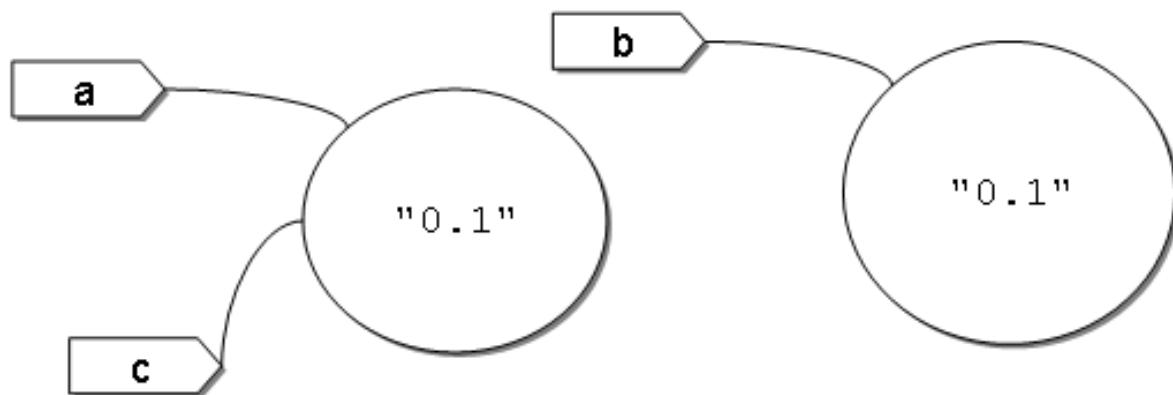
物件指定與相等性

```
BigDecimal a = new BigDecimal("0.1");  
BigDecimal b = new BigDecimal("0.1");  
System.out.println(a == b);           // 顯示 false  
System.out.println(a.equals(b));       // 顯示 true
```



物件指定與相等性

```
BigDecimal a = new BigDecimal("0.1");  
BigDecimal b = new BigDecimal("0.1");  
BigDecimal c = a;  
System.out.println(a == b);           // 顯示 false  
System.out.println(a == c);           // 顯示 true  
System.out.println(a.equals(b));      // 顯示 true
```



物件指定與相等性

- `==`用在物件型態，是比較兩個名稱是否參考同一物件，而`!=`正好相反，是比較兩個名稱是否沒參考同一物件
- 實際上，`equals()`可以自行定義如何比較兩物件的內含值

包裹基本型態

- 使用基本型態目的在於效率
- 然而更多時候，會使用類別建立實例，因為物件本身可以攜帶更多資訊
- 如果要讓基本型態像物件一樣操作，可以使用 **Long**、**Integer**、**Double**、**Float**、**Boolean**、**Byte** 等包裹器（Wrapper）類別來包裹（Wrap）基本型態

包裹基本型態

```
int data1 = 10;  
int data2 = 20;
```

```
Integer wrapper1 = new Integer(data1); ← ❶ 包裹基本型態  
Integer wrapper2 = new Integer(data2);
```

```
System.out.println(data1 / 3); ← ❷ 基本型態運算
```

```
System.out.println(wrapper1.doubleValue() / 3); ← ❸ 操作包裹器方法  
System.out.println(wrapper1.compareTo(wrapper2));
```

自動裝箱、拆箱

- 從J2SE 5.0之後提供了自動裝箱（Auto boxing）功能，可以如下包裹基本型態：

```
Integer wrapper = 10;
```

- 若使用自動裝箱功能來改寫一下IntegerDemo中的程式碼：

```
Integer data1 = 10;  
Integer data2 = 20;  
System.out.println(data1.doubleValue() / 3);  
System.out.println(data1.compareTo(data2));
```

自動裝箱、拆箱

- 自動裝箱運用的方法還可以如下：

```
int i = 10;  
Integer wrapper = i;
```

- 也可以使用更一般化的Number類別來自動裝箱，例如：

```
Number number = 3.14f;
```

自動裝箱、拆箱

- J2SE 5.0後可以自動裝箱，也可以自動拆箱（ Auto unboxing ）

```
Integer wrapper = 10;    // 自動裝箱  
int foo = wrapper;       // 自動拆箱
```

- 在運算時也可以進行自動裝箱與拆箱：

```
Integer i = 10;  
System.out.println(i + 10);  
System.out.println(i++);
```


自動裝箱、拆箱

- 再來看一個例子：

```
Boolean foo = true;  
System.out.println(foo && false);
```

裝箱的內幕

- 自動裝箱與拆箱的功能事實上是編譯器蜜糖 (Compiler sugar)

```
Integer i = 100;
```

```
Integer i = Integer.valueOf(100);
```

裝箱的內幕

- 例如下面的程式是可以通過編譯的：

```
Integer i = null;  
int j = i;
```

```
Integer integer = null;  
int i = integer.intValue();
```

 **NullPointerException**

裝箱的內幕

- 如果你如下撰寫，結果會是如何？

```
Integer i1 = 100;
Integer i2 = 100;
if (i1 == i2) {
    System.out.println("i1 == i2");
}
else {
    System.out.println("i1 != i2");
}
```

裝箱的內幕

- 如果你如下撰寫，結果會是如何？

```
Integer i1 = 200;
Integer i2 = 200;
if (i1 == i2) {
    System.out.println("i1 == i2");
}
else {
    System.out.println("i1 != i2");
}
```

裝箱的內幕

- 察看JDK資料夾src.zip中的java/lang資料夾中的Integer.java，你會看到valueOf()的實作內容：

```
public static Integer valueOf(int i) {  
    assert IntegerCache.high >= 127;  
    if (i >= IntegerCache.low && i <= IntegerCache.high)  
        return IntegerCache.cache[i + (-IntegerCache.low)];  
    return new Integer(i);  
}
```

- IntegerCache.low預設值是-128，
IntegerCache.high預設值是127

裝箱的內幕

- `IntegerCache.low`預設值是-128，執行時期無法更改
- `IntegerCache.high`預設值是127，可以於啟動JVM時，使用系統屬性

`java.lang.Integer.IntegerCache.high`來指定

```
> java -Djava.lang.Integer.IntegerCache.high=300 cc.openhome.Demo
```


裝箱的內幕

- 別使用`==`或`!=`來比較兩個物件實質內容值是否相同（因為`==`與`!=`是比較物件參考），而要使用`equals()`

```
Integer i1 = 200;
Integer i2 = 200;
if (i1.equals(i2)) {
    System.out.println("i1 == i2");
}
else {
    System.out.println("i1 != i2");
}
```

陣列基礎

- 若要用程式記錄Java小考成績，若有10名學生 ...

```
int score1 = 88;  
int score2 = 81;  
int score3 = 74;  
...  
int score10 = 93;
```

```
int[] scores = {88, 81, 74, 68, 78, 76, 77, 85, 95, 93};
```

陣列基礎

- 索引由0開始
- 如果存取超出索引範圍，就會拋出
ArrayIndexOutOfBoundsException

```
int[] scores = {88, 81, 74, 68, 78, 76, 77, 85, 95, 93};  
for(int i = 0; i < scores.length; i++) {  
    System.out.printf("學生分數：%d %n", scores[i]);  
}
```

陣列基礎

- 如果需求是循序地從頭至尾取出陣列值，從JDK5之後，有了更方便的增強式**for**迴圈（Enhanced for loop）語法

```
for(int score : scores) {  
    System.out.printf("學生分數：%d %n", score);  
}  
  
int ai[] = {88, 81, 74, 68, 78, 76, 77, 85, 95, 93};  
int ail[] = ai;  
int i = ail.length;  
for(int j = 0; j < i; j++) {  
    int k = ail[j];  
    ...  
}
```

陣列基礎

- 如果要設定值給陣列中某個元素，也是透過索引：

```
scores[3] = 86;  
System.out.println(scores[3]);
```

陣列基礎

• 宣告二維陣列

```
int[][] cords = {  
    {1, 2, 3},  
    {4, 5, 6}  
};  
for(int x = 0; x < cords.length; x++) {  
    for(int y = 0; y < cords[x].length; y++) {  
        System.out.printf("%2d", cords[x][y]);  
    }  
    System.out.println();  
}
```

➔ ❶ 宣告二維陣列並初始值

➔ ❷ 得知有幾列

➔ ❸ 取得每列的長度

➔ ❹ 指定列、行索引取得陣列元素

陣列基礎

- 可以用增強式for迴圈來改寫會比較簡潔：

```
for(int[] row : cords) {  
    for(int value : row) {  
        System.out.printf("%2d", value);  
    }  
    System.out.println();  
}
```


操作陣列物件

- 可以使用new關鍵字指定長度來建立陣列

```
int[] scores = new int[10];
```

資料型態	初始值
byte	0
short	0
int	0
long	0L
float	0.0F
double	0.0D
char	\u0000
boolean	false
類別	null

操作陣列物件

- 可以使用 **java.util.Arrays** 的 **fill()** 方法來設定新建陣列的元素值

```
int[] scores = new int[10];
for(int score : scores) {
    System.out.printf("%2d", score);
}
System.out.println();
Arrays.fill(scores, 60);
for(int score : scores) {
    System.out.printf("%3d", score);
}
```

操作陣列物件

- 想在new陣列時一併指定初始值

```
int[] scores = new int[] {88, 81, 74, 68, 78, 76, 77, 85, 95, 93};
```

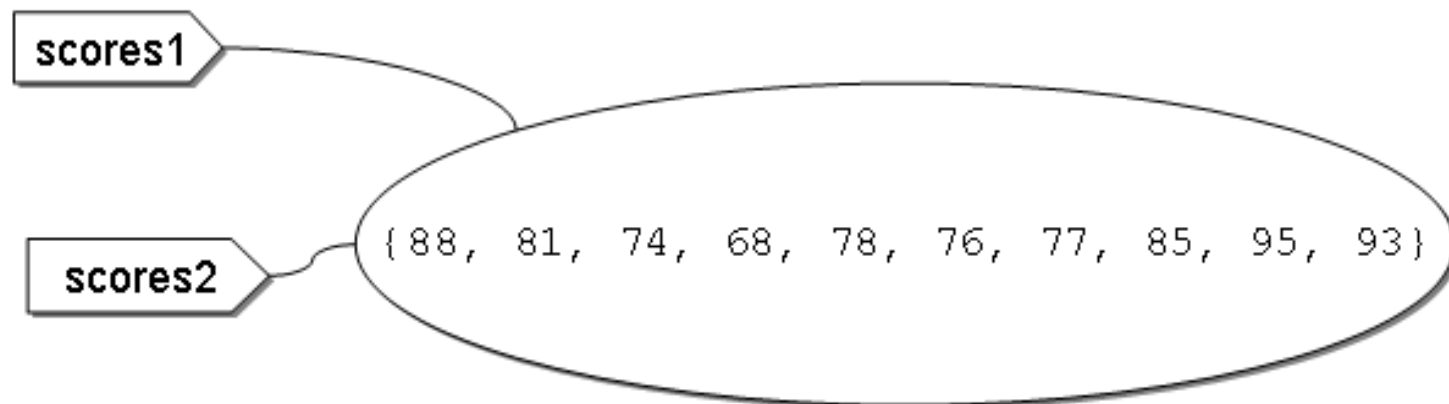
操作陣列物件

- 物件是根據類別而建立的實例，代表建立陣列物件的類別定義在哪？答案是由JVM動態產生。

操作陣列物件

- 看看以下這個片段會顯示什麼？

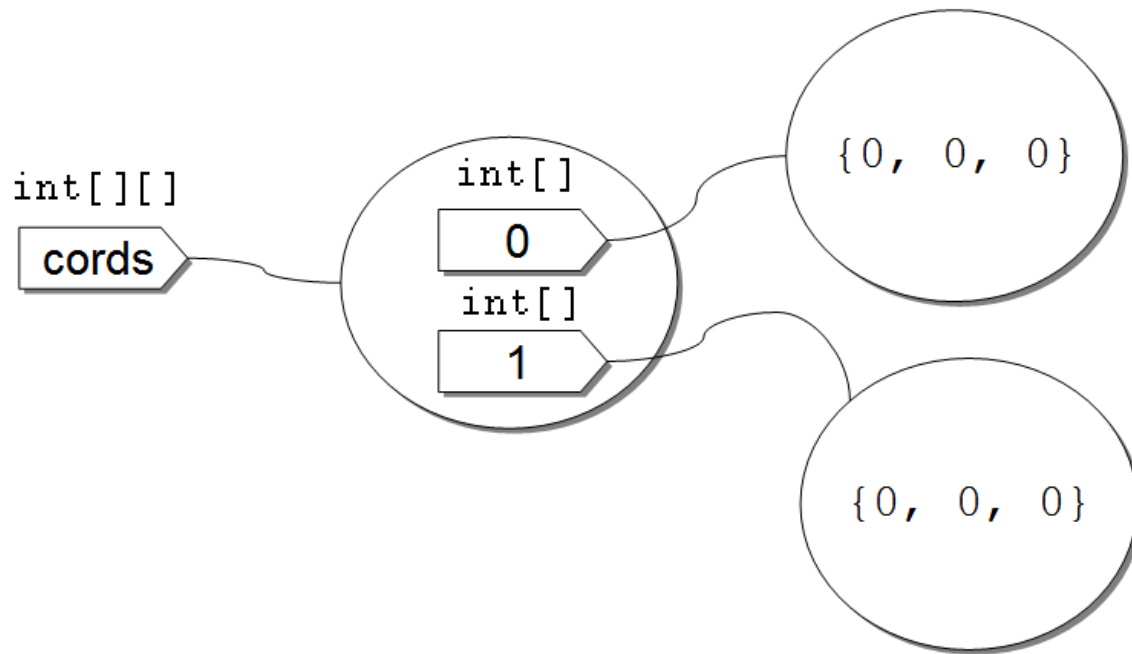
```
int[] scores1 = {88, 81, 74, 68, 78, 76, 77, 85, 95, 93};  
int[] scores2 = scores1;  
scores2[0] = 99;  
System.out.println(scores1[0]);
```



操作陣列物件

- 如果想用new建立二維陣列：

```
int[][] cords = new int[2][3];
```



操作陣列物件

- 應該可以知道為何要如下走訪二維陣列了：

```
for(int x = 0; x < cords.length; x++) {  
    for(int y = 0; y < cords[x].length; y++) {  
        System.out.printf("%2d", cords[x][y]);  
    }  
    System.out.println();  
}
```

操作陣列物件

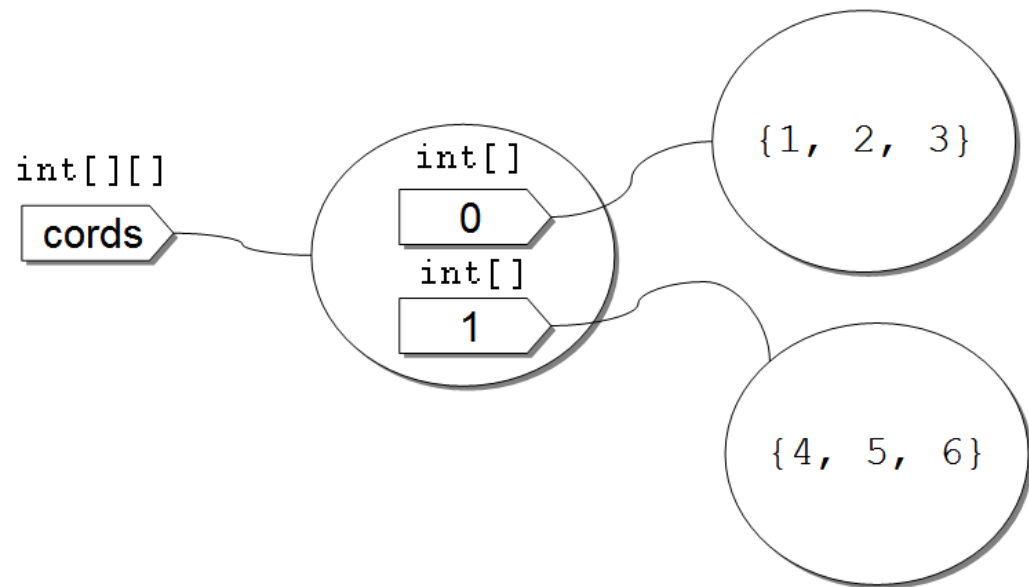
- 那麼這段增強式for語法是怎麼回事呢？

```
for(int[] row : cords) {  
    for(int value : row) {  
        System.out.printf("%2d", value);  
    }  
    System.out.println();  
}
```


操作陣列物件

- 如果使用new配置二維陣列後想要一併指定初值：

```
int[][] cords = new int[][] {  
    {1, 2, 3},  
    {4, 5, 6}  
};
```



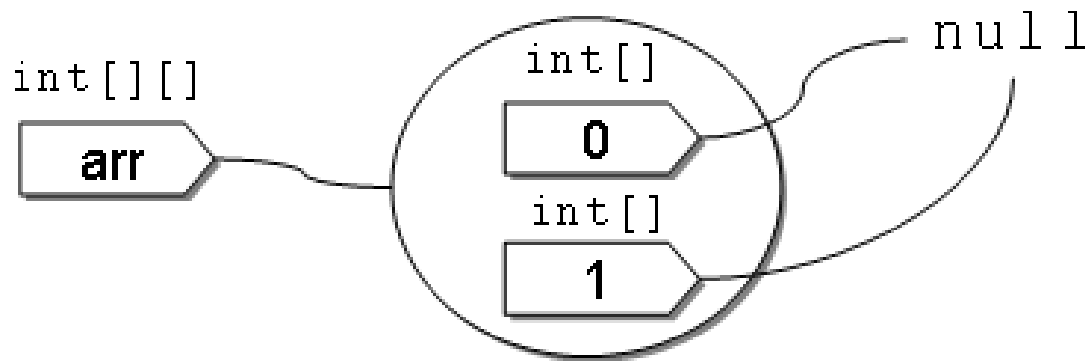
操作陣列物件

- 也可以建立不規則陣列：

```
int[][] arr = new int[2][]; ← ❶ 宣告 arr 參考的物件會有 2 個索引  
arr[0] = new int[] {1, 2, 3, 4, 5}; ← ❷ arr[0] 是長度為 5 的一維陣列  
arr[1] = new int[] {1, 2, 3}; ← ❸ arr[1] 是長度為 3 的一維陣列  
for(int[] row : arr) {  
    for(int value : row) {  
        System.out.printf("%2d", value);  
    }  
    System.out.println();  
}
```

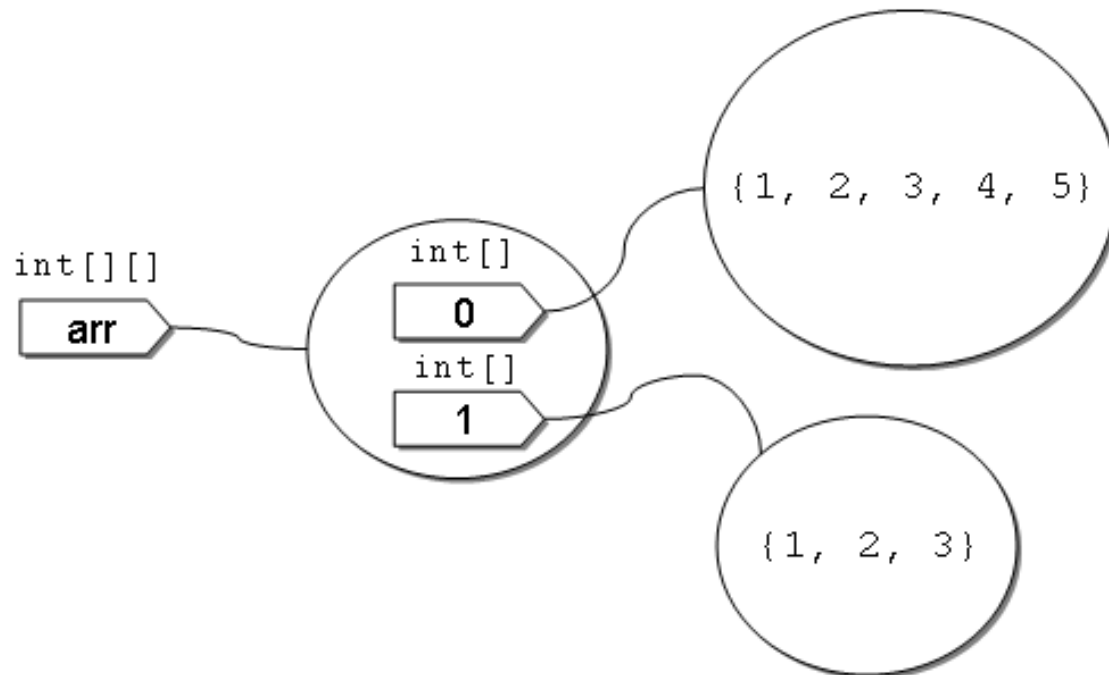
操作陣列物件

- 範例中 `new int[2][]` 僅提供第一個 `[]` 數值，這表示 `arr` 參考的物件會有兩個索引，但暫時參考至 `null` ❶



操作陣列物件

- 接著分別讓`arr[0]`參考至長度為5，而元素值為1、2、3、4、5的陣列，以及`arr[0]`參考至長度為3，而元素值為1、2、3的陣列



操作陣列物件

- 這麼建立陣列也是合法的：

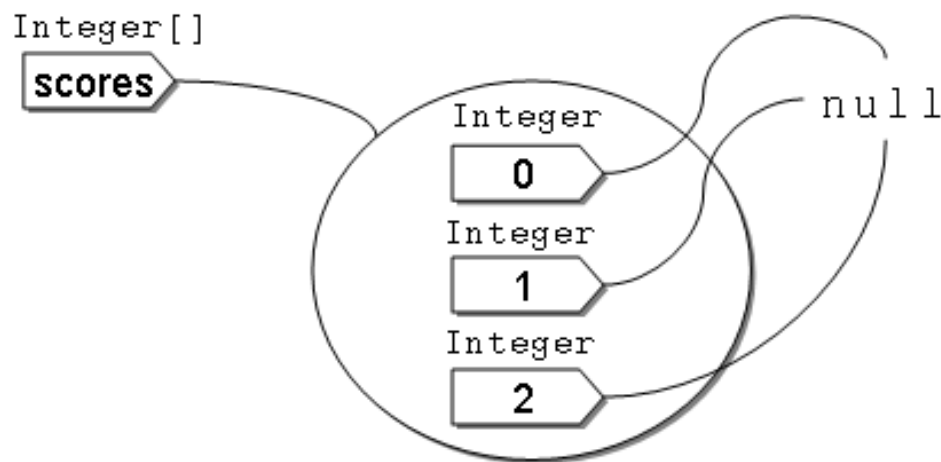
```
int[][] arr = {  
    {1, 2, 3, 4, 5},  
    {1, 2, 3}  
};
```

操作陣列物件

- 類別型態建立的陣列：

```
Integer[] scores = new Integer[3];
```

- 上面這個片段建立了幾個Integer物件呢？



操作陣列物件

- 每個索引其實都是Integer型態，可以讓你參考至Integer實例

```
Integer[] scores = new Integer[3];  
for(Integer score : scores) {  
    System.out.println(score);  
}  
  
scores[0] = new Integer(99);  
scores[1] = new Integer(87);  
scores[2] = new Integer(66);  
for(Integer score : scores) {  
    System.out.println(score);  
}
```

操作陣列物件

- 上面這個範例也可以結合自動裝箱語法

```
scores[0] = 99;  
scores[1] = 87;  
scores[2] = 66;
```


操作陣列物件

- 如果事先知道Integer陣列每個元素要放什麼，可以如下：

```
Integer[] scores = {new Integer(99), new Integer(87), new Integer(66)};
```

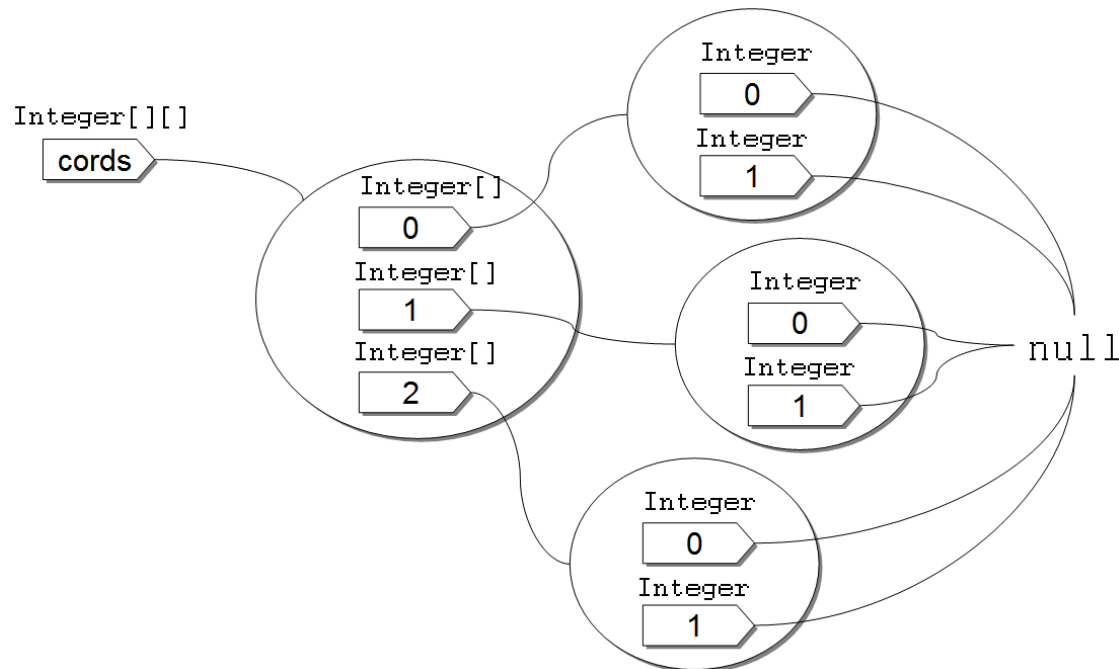
- 如果是JDK5以上，不結合自動裝箱來簡化程式撰寫，就有點可惜了

```
Integer[] scores = {99, 87, 66};
```

操作陣列物件

- 以下Integer二維陣列，建立了幾個Integer實例？

```
Integer[][] cords = new Integer[3][2];
```



陣列複製

- 以下這個並非陣列複製：

```
int[] scores1 = {88, 81, 74, 68, 78, 76, 77, 85, 95, 93};  
int[] scores2 = scores1;
```

- 要作陣列複製，基本作法是另行建立新陣列

```
int[] scores1 = {88, 81, 74, 68, 78, 76, 77, 85, 95, 93};  
int[] scores2 = new int[scores1.length];  
for(int i = 0; i < scores1.length; i++) {  
    scores2[i] = scores1[i];  
}
```

陣列複製

- 可以使用 **`System.arraycopy()`** 方法，這個方法會使用原生方式複製每個索引元素，比自行使用迴圈來得快：

```
int[] scores1 = {88, 81, 74, 68, 78, 76, 77, 85, 95, 93};  
int[] scores2 = new int[scores1.length];  
System.arraycopy(scores1, 0, scores2, 0, scores1.length);
```

陣列複製

- 如果使用JDK6以上，還有個更方便的 **`Arrays.copyOf()`** 方法，你不用另行建立新陣列，`Arrays.copyOf()` 會幫你建立

```
int[] scores1 = {88, 81, 74, 68, 78, 76, 77, 85, 95, 93};
int[] scores2 = Arrays.copyOf(scores1, scores1.length);
for(int score : scores2) {
    System.out.printf("%3d", score);
}
System.out.println();
scores2[0] = 99;
// 不影響 scores1 參考的陣列物件
for(int score : scores1) {
    System.out.printf("%3d", score);
}
```

陣列複製

- 事先建立的陣列長度不夠怎麼辦？那就只好建立新陣列，將原陣列內容複製至新陣列

```
int[] scores1 = {88, 81, 74, 68, 78, 76, 77, 85, 95, 93};
int[] scores2 = Arrays.copyOf(scores1, scores1.length * 2);
for(int score : scores2) {
    System.out.printf("%3d", score);
}
```

陣列複製

- 類別型態宣告的陣列則要注意參考的行為

```
class Clothes {
    String color;
    char size;
    Clothes(String color, char size) {
        this.color = color;
        this.size = size;
    }
}

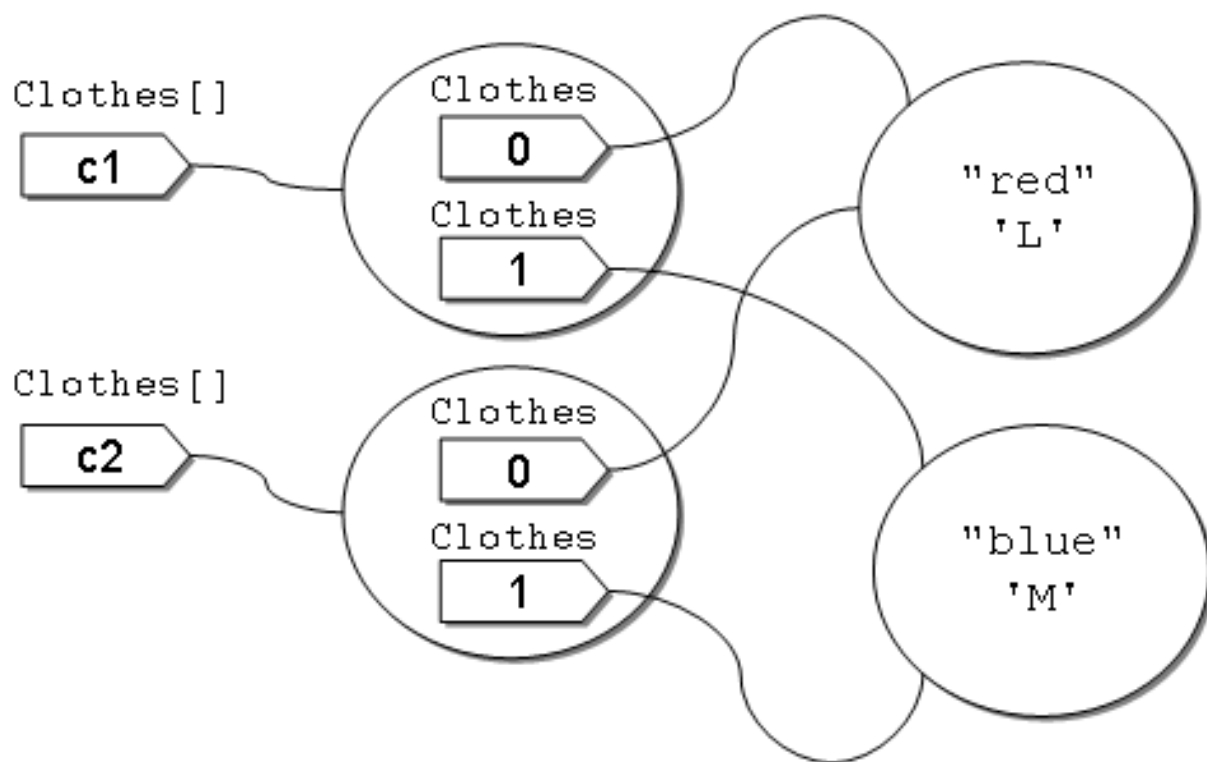
public class ShallowCopy {
    public static void main(String[] args) {
        Clothes[] c1 = {new Clothes("red", 'L'), new Clothes("blue", 'M')};
        Clothes[] c2 = new Clothes[c1.length];
        for(int i = 0; i < c1.length; i++) {
            c2[i] = c1[i];
        }
        c1[0].color = "yellow";
        System.out.println(c2[0].color);
    }
}
```

① 複製元素？

② 透過 c1 修改索引 0 物件

③ 透過 c2 取得索引 0 物件之顏色

陣列複製



陣列複製

- 實際上迴圈中僅將c1每個索引處所參考的物件，也給c2每個索引來參考，並沒有實際複製出Clothes物件
- 術語上來說，這叫作複製參考，或稱這個行為是淺層複製（Shallow copy）
- 無論是**`System.arraycopy()`**或**`Arrays.copyOf()`**，用在類別型態宣告的陣列時，都是執行淺層複製

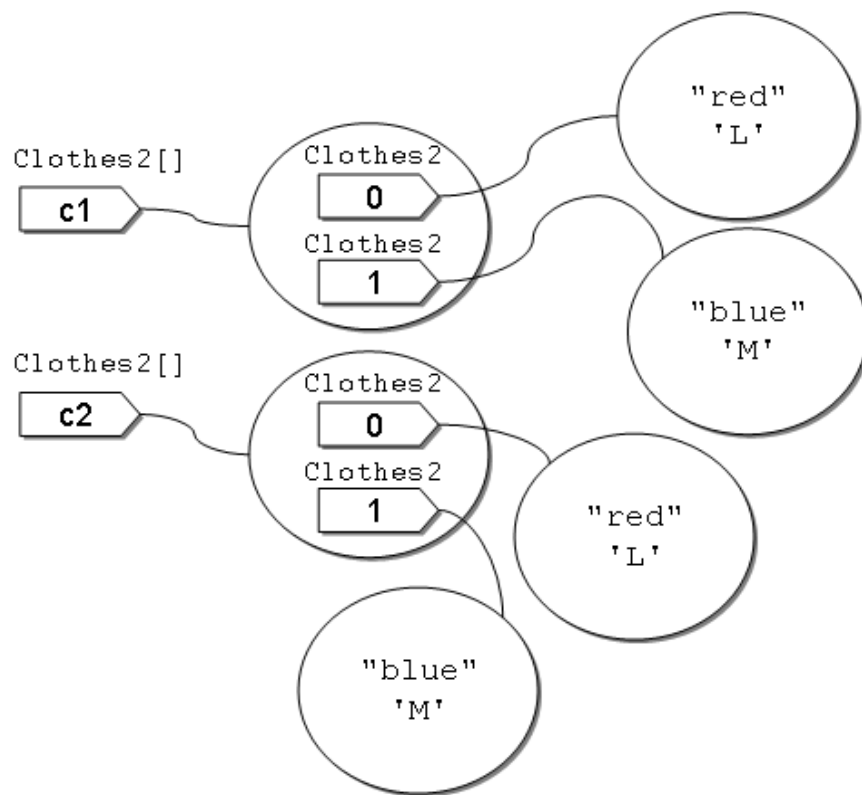
陣列複製

```
class Clothes2 {
    String color;
    char size;
    Clothes2(String color, char size) {
        this.color = color;
        this.size = size;
    }
}

public class DeepCopy {
    public static void main(String[] args) {
        Clothes2[] c1 = {new Clothes2("red", 'L'), new Clothes2("blue", 'M')};
        Clothes2[] c2 = new Clothes2[c1.length];
        for(int i = 0; i < c1.length; i++) {
            Clothes2 c = new Clothes2(c1[i].color, c1[i].size); ← 自行複製元素
            c2[i] = c;
        }
        c1[0].color = "yellow";
        System.out.println(c2[0].color);
    }
}
```

陣列複製

- 這個範例執行所謂深層複製 (Deep copy)



字串基礎

- 在Java中，字串是`java.lang.String`實例，用來包裹字元陣列
- 可以用`"`包括一串字元來建立字串

```
String name = "justin";           // 建立 String 實例
System.out.println(name);         // 顯示 justin
System.out.println(name.length()); // 顯示長度為 6
System.out.println(name.charAt(0)); // 顯示第一個字元 j
System.out.println(name.toUpperCase()); // 顯示 JUSTIN
```

字串基礎

- 已經有一個 `char[]` 陣列，也可以使用 `new` 來建構 `String` 實例

```
char[] cs = {'j', 'u', 's', 't', 'i', 'n'};  
String name = new String(cs);
```

- 也可以使用 `String` 的 **`toCharArray()`** 方法，以將字串以 `char[]` 陣列傳回：

```
char[] cs2 = name.toCharArray();
```

字串基礎

- 可以使用+運算來串接字串

```
String name = "Justin";  
System.out.println("你的名字是：" + name);
```

字串基礎

- 要將輸字串轉換為整數、浮點數等基本型態

方法	說明
<code>Byte.parseByte(number)</code>	將 <code>number</code> 剖析為 <code>byte</code> 整數
<code>Short.parseShort(number)</code>	將 <code>number</code> 為 <code>short</code> 整數
<code>Integer.parseInt(number)</code>	將 <code>number</code> 為 <code>int</code> 整數
<code>Long.parseLong(number)</code>	將 <code>number</code> 為 <code>long</code> 整數
<code>Float.parseFloat(number)</code>	將 <code>number</code> 為 <code>float</code> 浮點數
<code>Double.parseDouble(number)</code>	將 <code>number</code> 剖析為 <code>double</code> 浮點數

- 如果無法剖析傳入的 `String` 實例，則會拋出 **`NumberFormatException`**

字串基礎

```
Scanner scanner = new Scanner(System.in);
long sum = 0;
long number = 0;
do {
    System.out.print("輸入數字：");
    number = Long.parseLong(scanner.nextLine());
    sum += number;
} while (number != 0);
System.out.println("總合：" + sum);
```


字串基礎

- 程式進入點 `main()` 中的 `String[] args`
- 命令列引數 (Command line arguments)

```
> java cc.openhome.Average 1 2 3 4
```

```
public class Average {  
    public static void main(String[] args) {  
        long sum = 0;  
        for(String arg : args) {  
            sum += Long.parseLong(arg);  
        }  
        System.out.println("平均：" + (float) sum / args.length);  
    }  
}
```

字串特性

- Java字串有一些必須注意的特性：
 - 字串常量與字串池
 - 不可變動（Immutable）字串

字串特性

- 以下會顯示true或false？

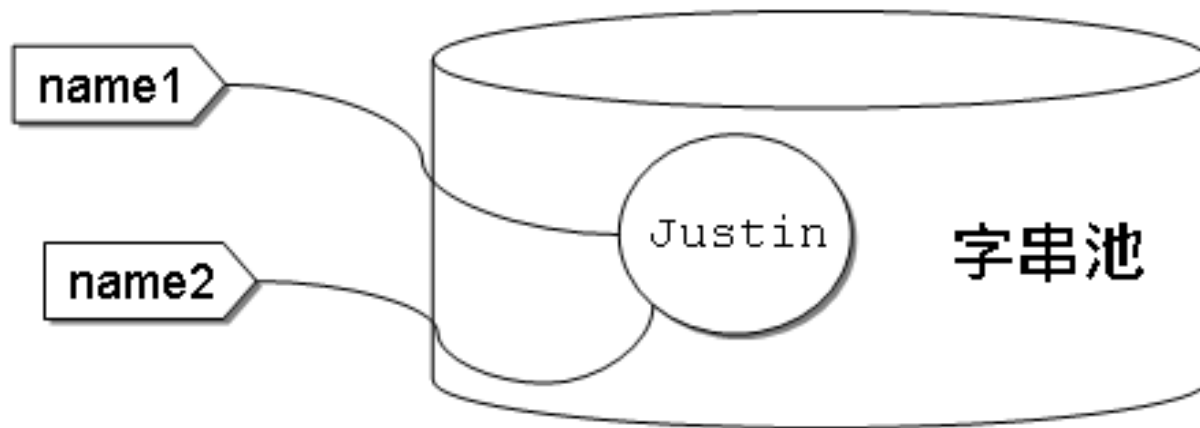
```
char[] name = {'J', 'u', 's', 't', 'i', 'n'};  
String name1 = new String(name);  
String name2 = new String(name);  
System.out.println(name1 == name2);
```

- 底下這個程式碼呢？

```
String name1 = "Justin";  
String name2 = "Justin";  
System.out.println(name1 == name2);
```

字串特性

- 以""包括的字串，只要內容相同（序列、大小寫相同），無論在程式碼中出現幾次，JVM都只會建立一個String實例，並在字串池（String pool）中維護

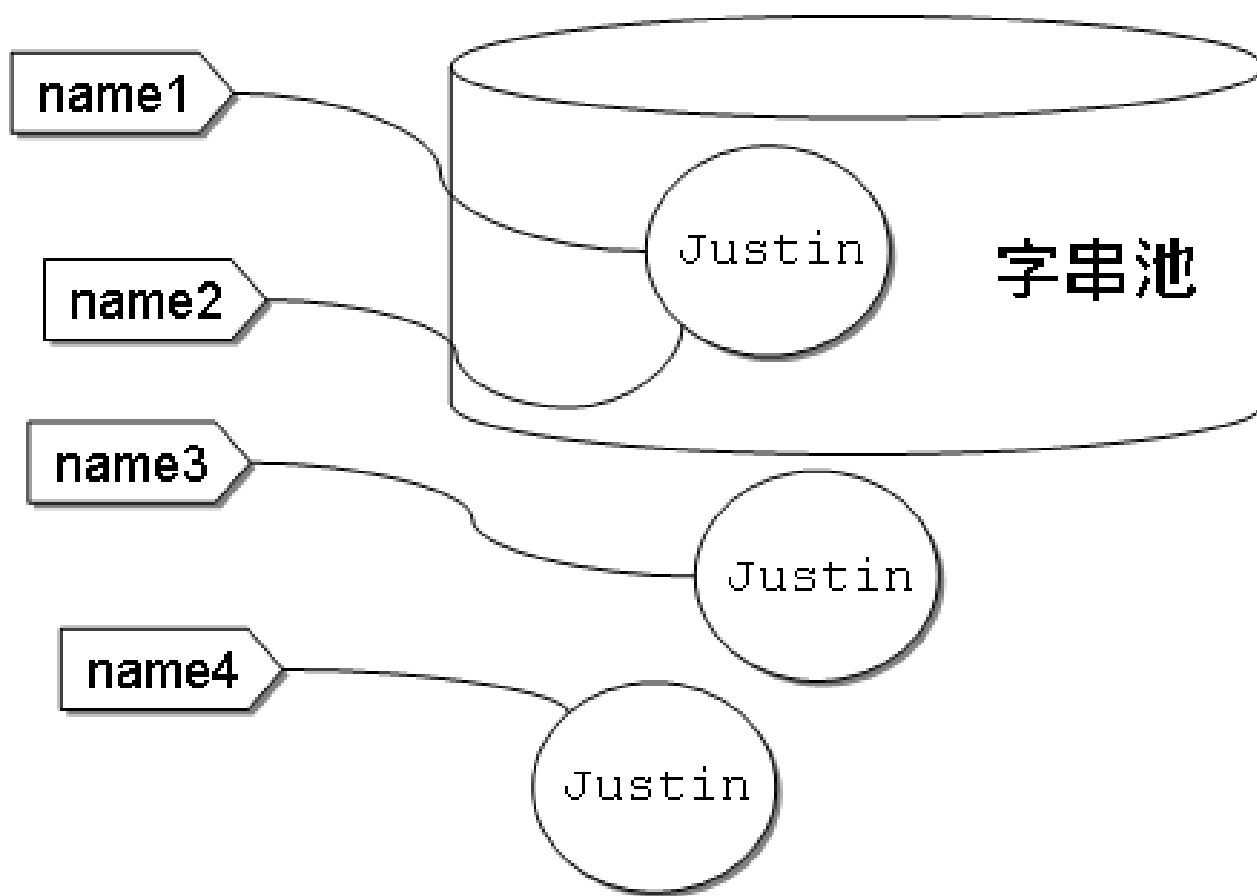


字串特性

- 用""寫下的字串稱為字串常量 (String literal)，既然你用"Justin"寫死了字串內容，基於節省記憶體考量，自然就不用為這些字串常量分別建立String實例

```
String name1 = "Justin";  
String name2 = "Justin";  
String name3 = new String("Justin");  
String name4 = new String("Justin");  
System.out.println(name1 == name2);  
System.out.println(name1 == name3);  
System.out.println(name3 == name4);
```

字串特性



字串特性

- 如果想比較字串實際字元內容是否相同，不要使用`==`，要使用`equals()`

```
String name1 = "Justin";  
String name2 = "Justin";  
String name3 = new String("Justin");  
String name4 = new String("Justin");  
System.out.println(name1.equals(name2));  
System.out.println(name1.equals(name3));  
System.out.println(name3.equals(name4));
```

字串特性

- 字串物件一旦建立，就無法更動物件內容
- 那麼使用+串接字串是怎麼達到的？

```
String name1 = "Java";  
String name2 = name1 + "World";  
System.out.println(name2);
```

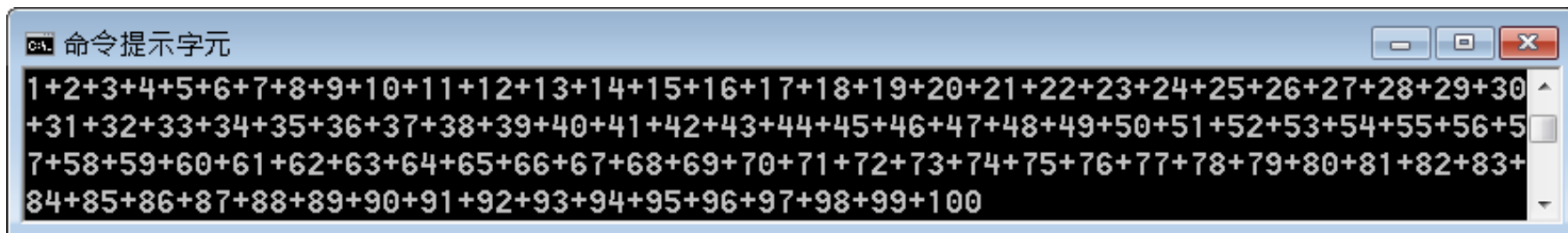
```
String s = "Java";  
String s1 = (new StringBuilder()).append(s).append("World").toString();  
System.out.println(s1);
```


字串特性

- 使用+串接字串會產生新的**String**實例
- 這並不是告訴你，不要使用+串接字串
- 只是在告訴你，不要將+用在重複性的串接場合，像是迴圈中或遞迴時使用+串接字串

字串特性

- 使用程式顯示下圖的結果，你會怎麼寫呢？



```
命令提示字元
1+2+3+4+5+6+7+8+9+10+11+12+13+14+15+16+17+18+19+20+21+22+23+24+25+26+27+28+29+30
+31+32+33+34+35+36+37+38+39+40+41+42+43+44+45+46+47+48+49+50+51+52+53+54+55+56+5
7+58+59+60+61+62+63+64+65+66+67+68+69+70+71+72+73+74+75+76+77+78+79+80+81+82+83+
84+85+86+87+88+89+90+91+92+93+94+95+96+97+98+99+100
```

字串特性

```
for(int i = 1; i < 101; i++) {  
    System.out.print(i);  
    if(i != 100) {  
        System.out.print('+');  
    }  
}
```

```
for(int i = 1; i < 100; i++) {  
    System.out.printf("%d+", i);  
}  
System.out.println(100);
```

```
String text = "";  
for(int i = 1; i < 100; i++) {  
    text = text + i + '+';  
}  
System.out.println(text + 100);
```

字串特性

```
public class OneTo100 {  
    public static void main(String[] args) {  
        StringBuilder builder = new StringBuilder();  
        for (int i = 1; i < 100; i++) {  
            builder.append(i).append('+');  
        }  
        System.out.println(builder.append(100).toString());  
    }  
}
```

字串特性

- 請問以下會顯示true或false？

```
String text1 = "Ja" + "va";  
String text2 = "Java";  
System.out.println(text1 == text2);
```

```
String s = "Java";  
String s1 = "Java";  
System.out.println(s == s1);
```

字串編碼

- 你寫的.java原始碼檔案是什麼編碼？
- 明明你的Windows純文字編輯器是Big5編碼，為什麼會寫下的字串在JVM中會是Unicode？

字串編碼

- 如果你的作業系統預設編碼是Big5，而你的文字編輯器是使用Big5編碼

```
public class Main {  
    public static void main(String[] args) {  
        System.out.println("Hello");  
        System.out.println("哈囉");  
    }  
}
```

```
> javac Main.java
```

字串編碼

- 產生的.class檔案，使用反組譯工具還原的程式碼中，會看到以下的內容：

```
public class Main {  
    public static void main(String args[]) {  
        System.out.println("Hello");  
        System.out.println("\u54C8\u56C9");  
    }  
}
```

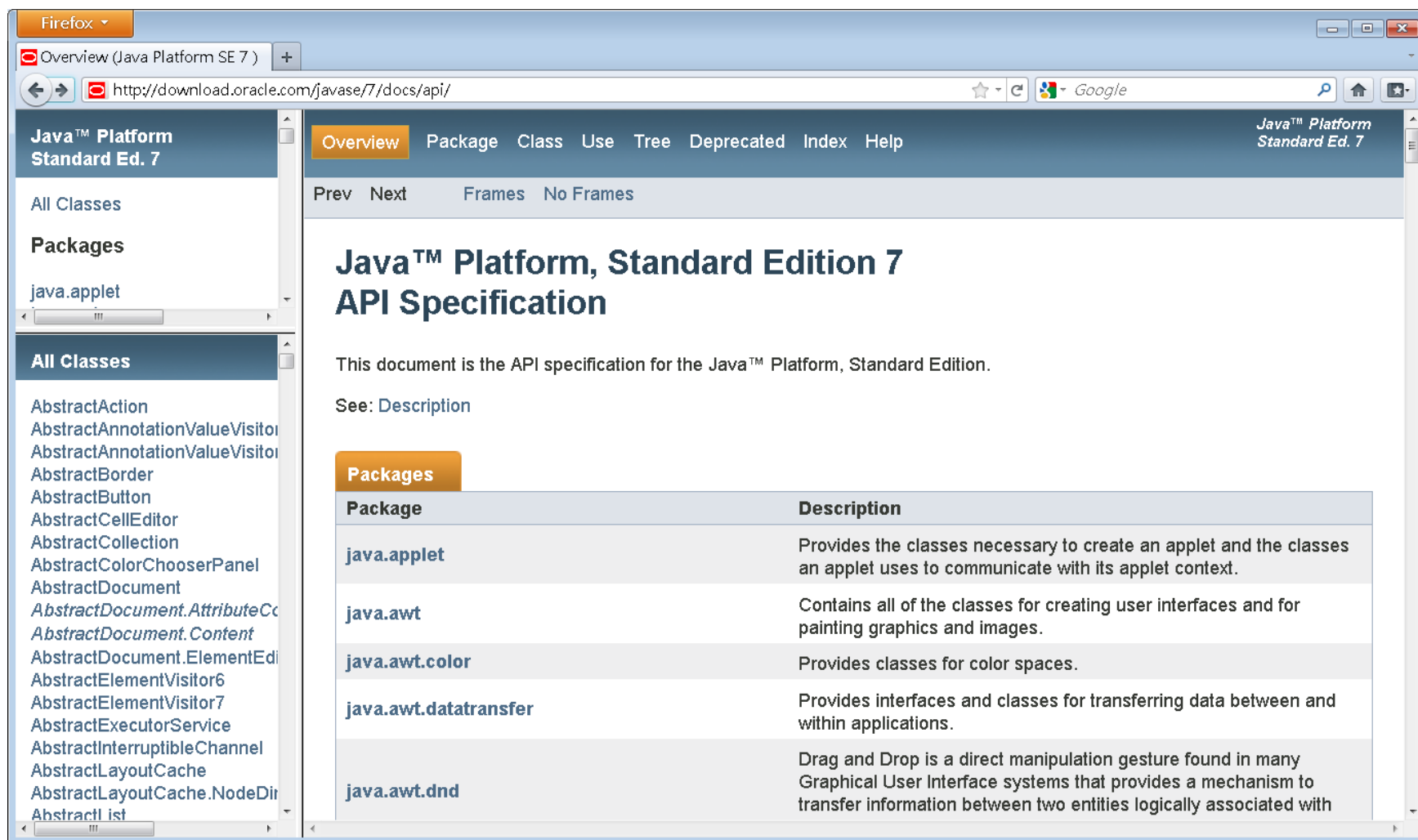
- JVM在載入.class之後，就是讀取Unicode編碼並產生對應的字串物件

字串編碼

- 使用javac指令沒有指定**-encoding**選項時，會使用作業系統預設編碼
- 如果你的文字編譯器是使用UTF-8編碼，那麼編譯時就要指定**-encoding**為UTF-8

```
> javac -encoding UTF-8 Main.java
```

查詢Java API文件



The screenshot shows a Firefox browser window displaying the Java Platform SE 7 API Specification page. The address bar shows the URL <http://download.oracle.com/javase/7/docs/api/>. The page title is "Java™ Platform, Standard Edition 7 API Specification".

The left sidebar contains a navigation menu with the following items:

- Java™ Platform Standard Ed. 7
- All Classes
- Packages
- java.applet
- All Classes
- AbstractAction
- AbstractAnnotationValueVisitor
- AbstractAnnotationValueVisitor
- AbstractBorder
- AbstractButton
- AbstractCellEditor
- AbstractCollection
- AbstractColorChooserPanel
- AbstractDocument
- AbstractDocument.AttributeCo
- AbstractDocument.Content
- AbstractDocument.ElementEdi
- AbstractElementVisitor6
- AbstractElementVisitor7
- AbstractExecutorService
- AbstractInterruptibleChannel
- AbstractLayoutCache
- AbstractLayoutCache.NodeDir
- AbstractList

The main content area shows the "Overview" tab selected. Below the navigation bar, there are links for "Prev", "Next", "Frames", and "No Frames". The title "Java™ Platform, Standard Edition 7 API Specification" is displayed. Below the title, a paragraph states: "This document is the API specification for the Java™ Platform, Standard Edition." A link "See: Description" is provided.

The "Packages" section is highlighted, showing a table of packages:

Package	Description
java.applet	Provides the classes necessary to create an applet and the classes an applet uses to communicate with its applet context.
java.awt	Contains all of the classes for creating user interfaces and for painting graphics and images.
java.awt.color	Provides classes for color spaces.
java.awt.datatransfer	Provides interfaces and classes for transferring data between and within applications.
java.awt.dnd	Drag and Drop is a direct manipulation gesture found in many Graphical User Interface systems that provides a mechanism to transfer information between two entities logically associated with

查詢Java API文件

The screenshot shows a Firefox browser window with the address bar displaying `http://download.oracle.com/javase/7/docs/api/`. The left sidebar contains a tree view of the Java API packages, with `String` selected under the `java.lang` package. The main content area displays the documentation for the `Class String`.

Class String

`java.lang.Object`
`java.lang.String`

All Implemented Interfaces:

`Serializable, CharSequence, Comparable<String>`

`public final class String`
`extends Object`
`implements Serializable, Comparable<String>, CharSequence`

The `String` class represents character strings. All string literals in Java programs, such as `"abc"`, are implemented as instances of this class.

Strings are constant; their values cannot be changed after they are created. String buffers support mutable strings. Because String objects are immutable they can be shared. For example:

```
String str = "abc";
```

is equivalent to:

```
char data[] = {'a', 'b', 'c'};  
String str = new String(data);
```

查詢Java API文件

Methods

Modifier and Type	Method and Description
char	<code>charAt(int index)</code> Returns the <code>char</code> value at the specified index.
int	<code>codePointAt(int index)</code> Returns the character (Unicode code point) at the specified index.
int	<code>codePointBefore(int index)</code> Returns the character (Unicode code point) before the specified index.
int	<code>codePointCount(int beginIndex, int endIndex)</code> Returns the number of Unicode code points in the specified text range of this <code>String</code> .
int	<code>compareTo(String anotherString)</code> Compares two strings lexicographically.
int	<code>compareToIgnoreCase(String str)</code> Compares two strings lexicographically, ignoring case differences.
String	<code>concat(String str)</code> Concatenates the specified string to the end of this string.
boolean	<code>contains(CharSequence s)</code> Returns true if and only if this string contains the specified sequence of char values.
boolean	<code>contentEquals(CharSequence cs)</code> Compares this string to the specified <code>CharSequence</code> .

查詢Java API文件

charAt

```
public char charAt(int index)
```

Returns the `char` value at the specified index. An index ranges from 0 to `length() - 1`. The first `char` value of the sequence is at index 0, the next at index 1, and so on, as for array indexing.

If the `char` value specified by the index is a [surrogate](#), the surrogate value is returned.

Specified by:

`charAt` in interface `CharSequence`

Parameters:

`index` - the index of the `char` value.

Returns:

the `char` value at the specified index of this string. The first `char` value is at index 0.

Throws:

[IndexOutOfBoundsException](#) - if the `index` argument is negative or not less than the length of this string.

查詢Java API文件

charAt

```
public char charAt(int index)
```

Returns the `char` value at the specified index. An index ranges from 0 to `length() - 1`. The first `char` value of the sequence is at index 0, the next at index 1, and so on, as for array indexing.

If the `char` value specified by the index is a [surrogate](#), the surrogate value is returned.

Specified by:

`charAt` in interface `CharSequence`

Parameters:

`index` - the index of the `char` value.

Returns:

the `char` value at the specified index of this string. The first `char` value is at index 0.

Throws:

[IndexOutOfBoundsException](#) - if the `index` argument is negative or not less than the length of this string.

查詢Java API文件

