基礎程式設計（Java）　講義

泛型類別與方法：型別參數、有界型別、萬用字元

Deitel & Deitel《Java How to Program: Early Objects》11e, Ch.20

§ 1　一、名詞速查表

中文	English	一句話定義	科目	章節
泛型	generics	把「型別」當成參數寫進類別或方法，同一份程式碼安全地套用到多種型別	Java	CH20
型別參數	type parameter	宣告在角括號裡的型別佔位符（慣例用 `T`、`E`、`K`、`V`）	Java	CH20
型別實參	type argument	使用時填進角括號的具體型別（如 `<Integer>`、`<String>`）	Java	CH20
泛型方法	generic method	在回傳型別前以 `<T>` 宣告型別參數的方法，可套用多種型別的引數	Java	CH20
泛型類別	generic class	類名後帶型別參數 `class Box<T>`，欄位／方法可用 `T`	Java	CH20
有界型別參數	bounded type parameter	用 `<T extends 上界>` 限制型別參數必須是某型別的子型別	Java	CH20
上界	upper bound	`extends` 後指定的型別，型別實參必須是它或它的子型別	Java	CH20
多型別參數	multiple type parameters	一個泛型同時宣告多個型別參數，如 `<K, V>`	Java	CH20
萬用字元	wildcard	角括號裡的 `?`，表示「某個未知型別」，多用於方法參數	Java	CH20
上界萬用字元	upper-bounded wildcard	`<? extends T>`，接受 T 或 T 的任一子型別（可讀不可寫）	Java	CH20
下界萬用字元	lower-bounded wildcard	`<? super T>`，接受 T 或 T 的任一父型別（可寫不可安全讀）	Java	CH20

中文	English	一句話定義	科目	章節
型別抹除	type erasure	編譯後型別參數被抹掉、換成上界（預設 `Object`），執行期不留泛型資訊	Java	CH20
原始型別	raw type	不帶角括號用泛型（如 `Box b`），失去型別檢查、會有警告，應避免	Java	CH20

§ 2 二、核心概念

CH16 已經在「用」泛型：`ArrayList<String>`、`Map<String, Integer>` 的角括號就是泛型。CH20 從「用別人寫好的泛型」進到「自己寫泛型」：把型別本身當成參數，讓同一份程式碼安全套用到多種型別。

> 全章最關鍵的一句話：**泛型把型別檢查從「執行期才爆」提前到「編譯期就擋」**。沒有泛型時，集合裡存的是 `Object`，取出來得自己轉型（cast），轉錯型別要到執行期才丟 `ClassCastException`；用了泛型，編譯器在編譯期就知道裡面裝什麼型別，放錯型別當場編譯錯，取出來也不必手動轉型。一句話：**型別安全（compile-time type safety）＋免手動轉型（no explicit casts）**。

第二條主線是**「一份取代多份」**：沒有泛型時，要寫「印 Integer 陣列」「印 Double 陣列」「印 String 陣列」三個幾乎一樣的多載方法；用泛型方法 `<T>` 只要寫一份，編譯器自動套用到任何型別。這也是泛型最直接的好處：消除為了不同型別重複貼上的程式碼。

§ 3 三、主要內容

3.1 1. 為何用泛型：型別安全 + 免轉型

沒有泛型的舊寫法，集合元素是 `Object`，取出要自己轉型，而且可以混入錯型別：

```
// 舊：原始型別，編譯器不檢查
java.util.ArrayList list = new java.util.ArrayList();
list.add("hello");
list.add(42);                       // 編譯器不擋，混進不同型別
String s = (String) list.get(1);    // 執行期才爆 ClassCastException
```

```
// 新：泛型，編譯期就擋
java.util.ArrayList<String> list = new java.util.ArrayList<>();
list.add("hello");
// list.add(42);                    // ← 編譯錯：型別不符（提前擋下）
String s = list.get(0);             // 不必轉型，編譯器知道是 String
```

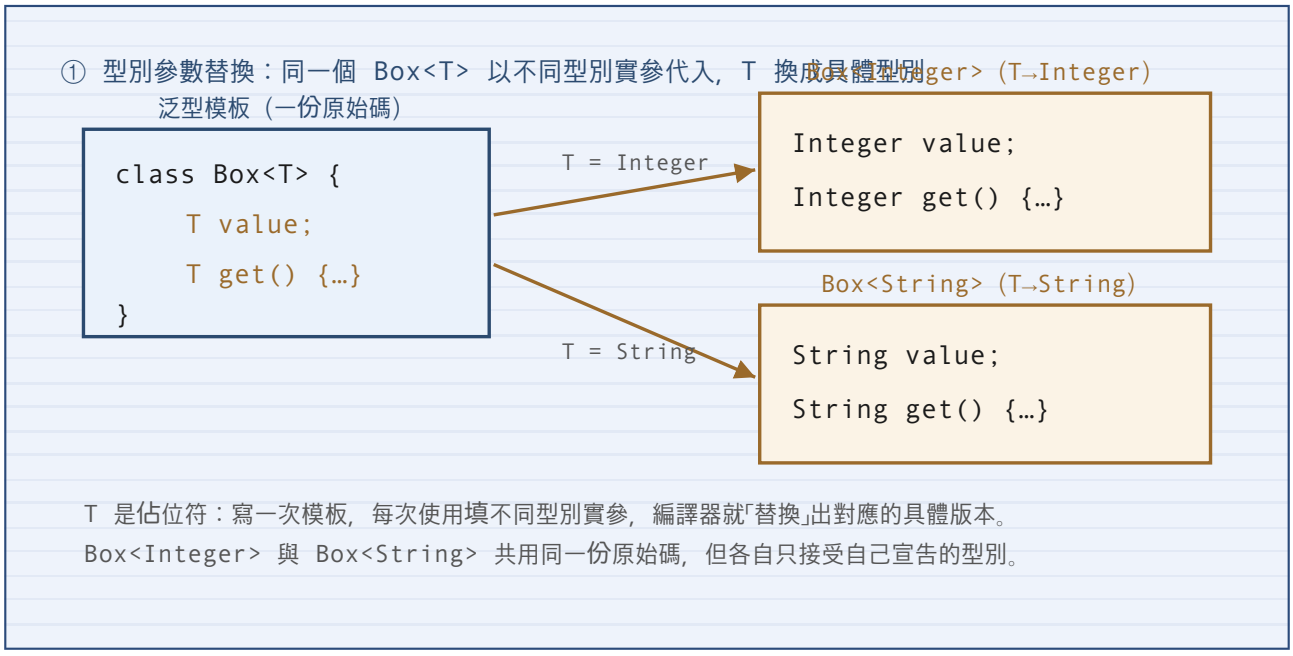
- **編譯期型別安全**：放錯型別當場編譯錯，不必等到執行期。
- **免手動轉型**：取出元素已是宣告的型別，省去 `(String)` 這類 cast，也避免轉錯。

3.2 2. 泛型方法：`<T>` 宣告在回傳型別前

把型別參數寫在**回傳型別之前**的角括號裡，方法內就能用這個型別參數。慣例用單一大寫字母：`T`（Type）、`E`（Element）、`K`（Key）、`V`（Value）。

```
public static <T> void printArray(T[] array) {    // <T> 在回傳型別 void 之前
    for (T element : array)                        // 方法內用 T 當型別
        System.out.printf("%s ", element);
    System.out.println();
}
// 一份方法，套用到任何型別的陣列：
Integer[] ints = {1, 2, 3};
String[] words = {"a", "b", "c"};
printArray(ints);     // T 自動推斷為 Integer
printArray(words);    // T 自動推斷為 String
```

- **型別參數宣告**：`<T>` 必須放在修飾子（`static`）之後、回傳型別之前。
- **型別實參**：呼叫時通常**不必明寫** `<Integer>`，編譯器由引數**自動推斷（type inference）**；必要時可寫 `MyClass.<Integer>printArray(ints)`。
- 一個泛型方法**取代了多個**只有型別不同的多載方法。



3.3 3. 型別參數與型別實參

- **型別參數（type parameter）**：定義泛型時寫的佔位符，如 `class Box<T>`、`<T> void f(...)` 裡的 `T`。
- **型別實參（type argument）**：使用泛型時填進去的具體型別，如 `Box<Integer>` 裡的 `Integer`。
- 型別實參**必須是參考型別**（類別、介面、陣列），**不能是基本型別**：寫 `Box<int>` 是編譯錯，要用包裝類別 `Box<Integer>`（自動裝箱會幫忙在 `int` 與 `Integer` 間轉換）。

```
Box<Integer> a = new Box<>(42);      // 型別實參 Integer（不是 int）
Box<String>  b = new Box<>("hi");    // 型別實參 String
// Box<int> c;                       // ← 編譯錯：基本型別不能當型別實參
```

3.4 4. 泛型類別：`class Box<T>`

類名後帶型別參數，類別內的欄位、方法參數、回傳型別都能用這個 `T`：

```
public class Box<T> {                  // T 是型別參數
    private T value;                   // 欄位型別是 T
    public Box(T value) { this.value = value; }
    public T get() { return value; }               // 回傳 T
    public void set(T value) { this.value = value; }
}
// Box<Integer> 的 get() 回 Integer；Box<String> 的 get() 回 String
Box<Integer> bi = new Box<>(100);
int n = bi.get();                      // 不必轉型；自動拆箱成 int
```

- 建立物件時右側可用**菱形運算子** `<>`（diamond）讓編譯器推斷：`new Box<>(100)` 等同 `new Box<Integer>(100)`。
- `static` **方法不能用類別的型別參數** `T`：`T` 屬於「物件層級」，`static` 方法屬類別、執行時可能沒有任何物件。要在 `static` 方法用泛型，得把它寫成獨立的泛型方法（自己宣告 `<T>`）。

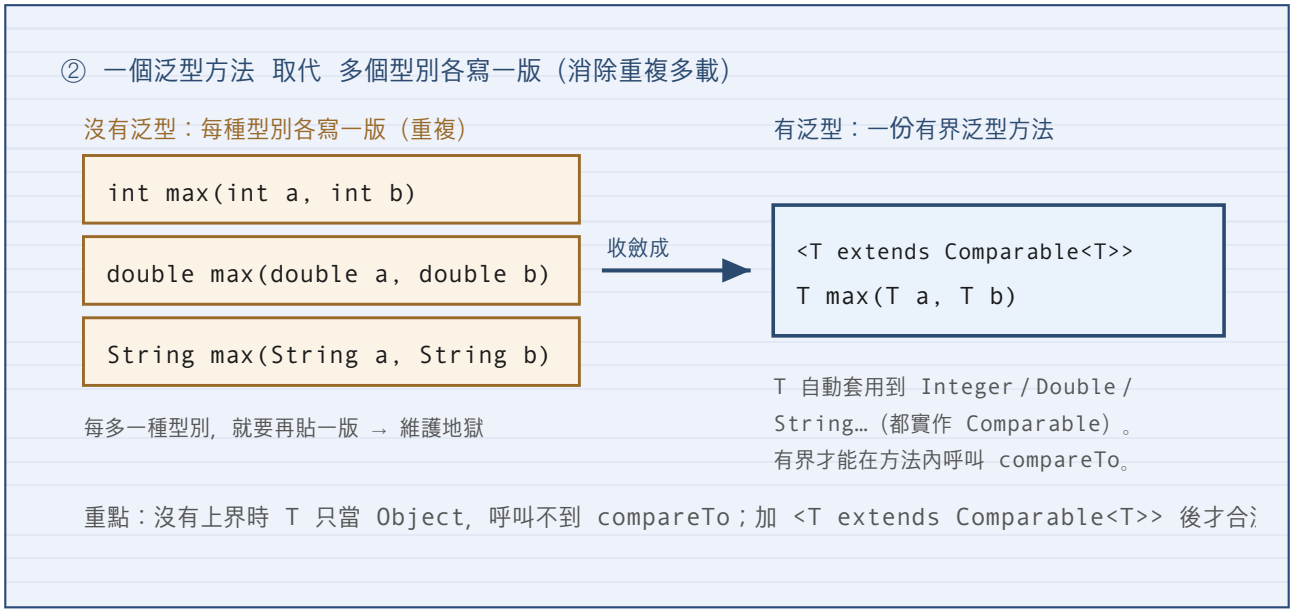
3.5 5. 有界型別參數：`<T extends Comparable<T>>`

預設型別參數可以是任何型別（上界是 `Object`），所以方法內只能呼叫 `Object` 有的方法。若方法內要呼叫例如 `compareTo`，必須**限制** `T` **一定實作** `Comparable`，這叫**有界型別參數**：

```
public static <T extends Comparable<T>> T max(T a, T b) {   // T 必須可比較
    return a.compareTo(b) >= 0 ? a : b;     // 因為有界，才能呼叫 compareTo
}
max(3, 7);            // T = Integer（Integer 實作 Comparable）→ 7
max("apple", "bee");  // T = String（String 實作 Comparable）→ "bee"
```

- `extends` 在泛型裡同時涵蓋「類別的 extends」與「介面的 implements」，一律寫 `extends`。

- **上界**就是 `extends` 後那個型別；型別實參必須是它或它的子型別。沒寫上界時，上界預設是 `Object`。
- 多重上界用 `&`：`<T extends Number & Comparable<T>>`。



3.6 6. 多型別參數：`<K, V>`

一個泛型可以同時有多個型別參數，用逗號分隔。最常見是「鍵—值」配對 `<K, V>`（Key／Value）：

```
public class Pair<K, V> {              // 兩個型別參數
    private K key;
    private V value;
    public Pair(K key, V value) { this.key = key; this.value = value; }
    public K getKey() { return key; }
    public V getValue() { return value; }
}
Pair<String, Integer> p = new Pair<>("age", 18);   // K=String, V=Integer
System.out.println(p.getKey() + " = " + p.getValue());   // age = 18
```

- 各型別參數各自獨立，`K` 與 `V` 可填不同型別，也可填相同型別。
- `java.util.Map<K, V>`（CH16）正是用兩個型別參數的標準例子。

3.7 7. 萬用字元：`<?>`／`<? extends>`／`<? super>`

寫**接受泛型的方法參數**時，常需要「某個未知型別的集合」。`List<Number>` 不接受 `List<Integer>`（泛型不具型別協變），這時用**萬用字元** `?`：

```
// 無界：任何型別的 List 都收（只能當 Object 讀、不能 add 元素）
public static void printAll(java.util.List<?> list) {
    for (Object o : list) System.out.print(o + " ");
}

// 上界 <? extends Number>：Number 或其子型別（Integer/Double…）。可「讀」，不可「寫」
public static double sum(java.util.List<? extends Number> list) {
    double total = 0;
    for (Number n : list) total += n.doubleValue();   // 安全地當 Number 讀
    return total;                                      // list.add(...) 會編譯錯
}
```

- `<?>` **無界萬用字元**：任何型別都收；元素只能當 `Object` 看，**不能** `add`（除了 `null`）。
- `<? extends T>` **上界萬用字元**：T 或其子型別，**適合「讀」**（生產者）；不能往裡加元素。
- `<? super T>` **下界萬用字元**：T 或其父型別，**適合「寫」**（消費者）；可 `add` T 及其子型別，讀出來只保證是 `Object`。
- 口訣 **PECS**：Producer-Extends, Consumer-Super（要讀用 `extends`、要寫用 `super`）。

3.8 8. 型別抹除（type erasure，了解即可）

Java 泛型是**編譯期**機制：編譯後型別參數被「抹掉」，換成它的上界（沒上界就換成 `Object`），執行期不保留泛型資訊。

```
// 你寫的：
Box<Integer> bi = new Box<>(1);
Box<String>  bs = new Box<>("a");
// 編譯後（概念上）：T 都被抹成 Object，bi 與 bs 的「類別」其實是同一個 Box
System.out.println(bi.getClass() == bs.getClass());   // true（都是 Box.class）
```

- 後果：執行期分不出 `Box<Integer>` 與 `Box<String>`；不能寫 `new T[]`、`T.class`、`instanceof Box<Integer>`（這些需要執行期的型別資訊）。
- 好處：與沒有泛型的舊程式碼相容（泛型是 Java 5 才加入）。
- early-objects 階段知道「泛型只在編譯期檢查、執行期被抹除」即可，細節（橋接方法等）不在考試範圍。

§ 4 四、語法與 API 速查

```
// 泛型方法：<T> 在修飾子之後、回傳型別之前
public static <T> void printArray(T[] arr) { ... }
ClassName.<Integer>printArray(ints);      // 明寫型別實參（通常可省，靠推斷）

// 泛型類別：類名後帶型別參數
public class Box<T> { private T value; public T get(){return value;} }
Box<Integer> b = new Box<>(42);           // 菱形 <> 推斷型別實參

// 多型別參數
public class Pair<K, V> { ... }
Pair<String, Integer> p = new Pair<>("age", 18);

// 有界型別參數（上界）：型別實參必須是 Comparable 的子型別
public static <T extends Comparable<T>> T max(T a, T b) { ... }
<T extends Number & Comparable<T>>        // 多重上界用 &

// 萬用字元
List<?> l;                                // 無界：任何型別，只能讀為 Object、不能 add
List<? extends Number> r;                 // 上界：讀（producer）；不能 add
List<? super Integer> w;                  // 下界：寫（consumer）；可 add Integer

// 型別實參規則
Box<Integer> ok;      // 參考型別（包裝類別）
// Box<int> bad;      // ← 編譯錯：不能用基本型別
```

- **慣例字母**：`T` 型別、`E` 元素、`K` 鍵、`V` 值、`N` 數值。
- **PECS**：要讀（producer）用 `<? extends T>`；要寫（consumer）用 `<? super T>`。
- **泛型 vs Object**：泛型在編譯期擋型別錯、取出免轉型；用 `Object` 要手動轉型、轉錯到執行期才爆。

§ 5 五、常見錯誤

- **型別實參用基本型別**：`Box<int>`、`List<double>` 編譯錯；要用包裝類別 `Box<Integer>`、`List<Double>`。
- **沒加上界就呼叫 `compareTo`／其他方法**：型別參數無上界時上界是 `Object`，方法內只能用 `Object` 的方法；要呼叫 `compareTo` 必須寫 `<T extends Comparable<T>>`。
- **以為 `List<Integer>` 是 `List<Number>` 的子型別**：泛型**不協變**，`List<Integer>` 不能傳給收 `List<Number>` 的參數；要相容用 `List<? extends Number>`。
- **對 `<? extends T>` 集合 `add` 元素**：上界萬用字元只能讀不能寫，`list.add(x)` 會編譯錯（除了 `null`）。
- **在 `static` 方法用類別的型別參數 `T`**：類別的 `T` 屬物件層級，`static` 方法用不到；要寫成自己宣告 `<T>` 的泛型方法。

- **執行期想拿泛型型別**：`new T[]`、`T.class`、`o instanceof List<String>` 都不行——型別抹除後執行期沒有 `T` 的資訊。
- **用原始型別（raw type）**：寫 `List list = new ArrayList();`（不帶角括號）會失去型別檢查、編譯器給「unchecked」警告；一律帶型別實參。
- **角括號裡用萬用字元 `?` 當「可寫任意型別」**：`List<?>` 不能 `add`（除了 `null`），它表示「未知的單一型別」而非「任意型別都能放」。

§ 6 六、練習題

例題 1：泛型方法印任意型別陣列

寫一個泛型方法 ` void printArray(T[] a)`，用增強 for 印出陣列每個元素（空白分隔、結尾換行）。在 `main` 用 `Integer[]` 與 `String[]` 各呼叫一次。

> 1. `` 寫在 `static` 之後、回傳型別 `void` 之前 2. 方法內 `for (T e : a) System.out.printf("%s ", e);` 後 `System.out.println();` 3. 呼叫時不必明寫型別實參，編譯器自動推斷

```
public class PrintArrayDemo {
    public static <T> void printArray(T[] a) {
        for (T e : a) System.out.printf("%s ", e);
        System.out.println();
    }
    public static void main(String[] args) {
        Integer[] ints = {1, 2, 3};
        String[] words = {"a", "b", "c"};
        printArray(ints);    // 1 2 3
        printArray(words);   // a b c
    }
}
```

> 易錯：`` 漏寫或放錯位置（必須在回傳型別前）；用基本型別 `int[]`（泛型方法的陣列元素要參考型別）。

例題 2：泛型類別 Pair<K, V>

設計泛型類別 `Pair`，含 `private K key; private V value;`、建構子與 `getKey()`／`getValue()`。在 `main` 建一個 `Pair("age", 18)` 並印出 `age = 18`。

> 1. 類名後 `<K, V>` 兩個型別參數 2. 兩欄位型別分別是 `K`、`V`，getter 回對應型別 3. `new Pair<>("age", 18)` 用菱形推斷

```
public class Pair<K, V> {
    private K key;
    private V value;
    public Pair(K key, V value) { this.key = key; this.value = value; }
    public K getKey() { return key; }
    public V getValue() { return value; }
    public static void main(String[] args) {
        Pair<String, Integer> p = new Pair<>("age", 18);
        System.out.println(p.getKey() + " = " + p.getValue());   // age = 18
    }
}
```

> 易錯：型別參數只宣告一個卻用兩個；建構子參數型別寫成具體型別而非 `K`／`V`。

例題 3：有界型別參數找最大

寫泛型方法 `> T max(T a, T b, T c)` 回傳三者最大。在 `main` 對 `Integer` 與 `String` 各呼叫一次。

> 1. 上界 `>` 才能在方法內呼叫 `compareTo` 2. 先比 a、b 取較大，再與 c 比 3. `max(3, 7, 5)` → 7；`max("apple", "bee", "cat")` → "cat"

```
public class MaxDemo {
    public static <T extends Comparable<T>> T max(T a, T b, T c) {
        T m = a;
        if (b.compareTo(m) > 0) m = b;
        if (c.compareTo(m) > 0) m = c;
        return m;
    }
    public static void main(String[] args) {
        System.out.println(max(3, 7, 5));                // 7
        System.out.println(max("apple", "bee", "cat")); // cat
    }
}
```

> 易錯：漏 `extends Comparable` 上界（呼叫不到 compareTo，編譯錯）；用 `>` 比物件（物件要用 compareTo）。

例題 4：泛型 Box 存取

設計泛型類別 `Box`，含 `private T value`、建構子、`get()`／`set(T)`。在 `main` 建 `Box` 與 `Box`，各 set 一個值再 get 印出。

> 1. `class Box`，欄位 `private T value;` 2. `get()` 回 `T`、`set(T value)` 設值 3. `Box bi = new Box<>(0); bi.set(42);`

```
public class Box<T> {
    private T value;
    public Box(T value) { this.value = value; }
    public T get() { return value; }
    public void set(T value) { this.value = value; }
    public static void main(String[] args) {
        Box<Integer> bi = new Box<>(0);
        bi.set(42);
        System.out.println("Integer Box: " + bi.get());   // 42
        Box<String> bs = new Box<>("");
        bs.set("hello");
        System.out.println("String Box: " + bs.get());    // hello
    }
}
```

易錯：把 `value` 宣告成 `Object` 而非 `T`（失去型別安全）；`Box`（基本型別不能當型別實參）。

例題 5：泛型 swap 交換陣列兩元素

寫泛型方法 ` void swap(T[] a, int i, int j)` 交換陣列索引 i、j 的兩元素。在 `main` 用 `String[]` 交換第 0 與第 2 個後印出整個陣列。

> 1. `` 宣告在回傳型別前 2. 用暫存 `T tmp = a[i]; a[i] = a[j]; a[j] = tmp;` 3. 印出用 `java.util.Arrays.toString(a)`

```
public class SwapDemo {
    public static <T> void swap(T[] a, int i, int j) {
        T tmp = a[i];
        a[i] = a[j];
        a[j] = tmp;
    }
    public static void main(String[] args) {
        String[] words = {"first", "second", "third"};
        swap(words, 0, 2);
        System.out.println(java.util.Arrays.toString(words));   // [third,
second, first]
    }
}
```

> 易錯：暫存變數用具體型別而非 `T`；忘了泛型陣列元素須為參考型別（不能 `int[]`）。

§ 7 七、自我檢核

- [ ] 能說出泛型的兩個好處：編譯期型別安全、取出免手動轉型。
- [ ] 會寫泛型方法，知道 `<T>` 要放在修飾子之後、回傳型別之前。
- [ ] 分得清型別參數（定義時的佔位符）與型別實參（使用時填的具體型別），知道型別實參不能是基本型別。
- [ ] 會寫泛型類別 `class Box<T>` 與多型別參數 `<K, V>`，會用菱形 `<>`。
- [ ] 會用有界型別參數 `<T extends Comparable<T>>`，知道沒上界時只能用 `Object` 的方法。
- [ ] 知道萬用字元 `<?>`／`<? extends>`／`<? super>` 的差別與 PECS 口訣。
- [ ] 知道泛型是編譯期機制、執行期型別被抹除（type erasure），故不能 `new T[]`、`T.class`。