Java Generic Programming

جامعة طرابلس - كلية تقنية المعلومات

د. عبدالحميد الواعر

Java Generic Programming

مثال:

```java
public class PrintArrayExample {
    public static void main(String[] args) {
      // create arrays of Integer, Double and Character
      Integer[] integerArray = {1, 2, 3, 4, 5, 6};
      Double[] doubleArray = {1.1, 2.2, 3.3, 4.4, 5.5, 6.6, 7.7};
      Character[] characterArray = {'H', 'E', 'L', 'L', 'O'};

      printArray(integerArray); // pass an Integer array
      printArray(doubleArray); // pass a Double array
      printArray(characterArray); // pass a Character array
    }

    // method printArray to print Integer array
    public static void printArray(Integer[] inputArray){
        for(int i=0;i<inputArray.length;i++)
            System.out.printf("%s ",inputArray[i]);
        System.out.println();
    }

    // method printArray to print Double array
    public static void printArray(Double[] inputArray){
        for(int i=0;i<inputArray.length;i++)
            System.out.printf("%s ",inputArray[i]);
        System.out.println();
    }

    // method printArray to print Character array
    public static void printArray(Character[] inputArray){
        for(int i=0;i<inputArray.length;i++)
            System.out.printf("%s ",inputArray[i]);
        System.out.println();
    }
}
```

مثال:

```java
public class PrintArrayGPExample {
    public static void main(String[] args) {
      // create arrays of Integer, Double and Character
      Integer[] integerArray = {1, 2, 3, 4, 5, 6};
      Double[] doubleArray = {1.1, 2.2, 3.3, 4.4, 5.5, 6.6, 7.7};
      Character[] characterArray = {'H', 'E', 'L', 'L', 'O'};

      printArray(integerArray); // pass an Integer array
      printArray(doubleArray); // pass a Double array
      printArray(characterArray); // pass a Character array
    }

    // method printArray to print an array
    public  static <T> void printArray(T[] inputArray){
        for(int i=0;i<inputArray.length;i++)
            System.out.printf("%s ",inputArray[i]);
        System.out.println();
    }

}
```

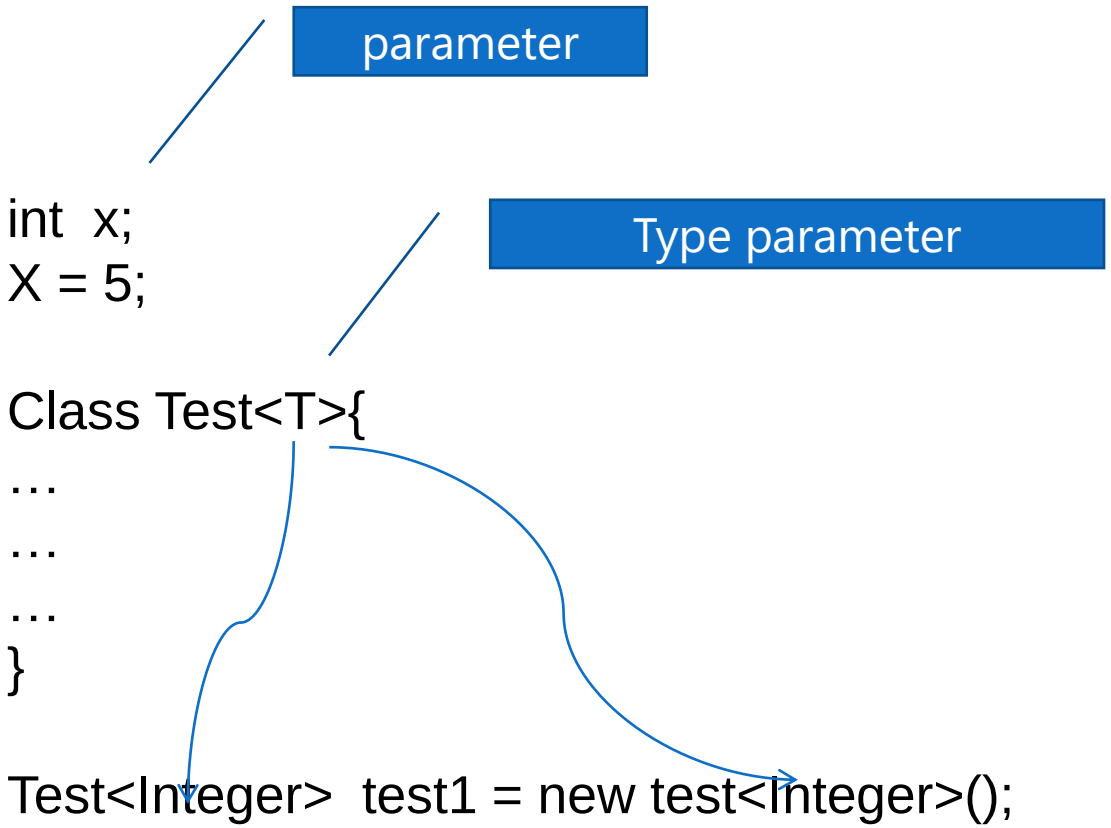
Java generic programming

Generic Programming

- هي طريقة في البرمجة تسمح بأنشاء class, interface, method يكون فيها أنواع البيانات المستخدمة على هيئة parameters تعرف ب (type parameters)يمكن تحديدها وقت الاستخدام مما يسمح بأعاده استخدامهم مع أنواع مختلفة من البيانات.

- type parameters شبيهه ب parameters المستخدمة عند الاعلان عن methods والفرق فقط في أن parameters تقبل قيم بينما type parameters تقبل Reference Type .

مثال:


parameter
Type parameter
int x;
X = 5;
Class Test<T>{
…
…
…
}
Test<Integer> test1 = new test<Integer>();

Java generic programming

- استخدام Generic Programming يقوم بعمل type check at compile time مما يحد من حدوث الاخطاء.
- يحد من عملية cast للبيانات .

مثال:

```java
package java.lang;

public interface Comparable {
  public int compareTo(Object o)
}
```

(a) Prior to JDK 1.5

```java
package java.lang;

public interface Comparable<T> {
  public int compareTo(T o)
}
```

(b) Since JDK 1.5

```java
Comparable c = new Date();
System.out.println(c.compareTo("red"));
```

(a) Prior to JDK 1.5

```java
Comparable<Date> c = new Date();
System.out.println(c.compareTo("red"));
```

(b) Since JDK 1.5

Runtime Error

Compile time Error

مثال:

```
ArrayList<String> list = new ArrayList<String>();
```

```
list.add("Red");
```

OK

```
list.add(new Integer(1));
```

Compile time error

مثال:

```
1  ArrayList<String> list = new ArrayList<String>();
2  list.add("Red");
3  list.add("White");
4  String s = list.get(0); // No casting is needed
```

قبل نسخة جافا 1.5 كانت هناك الحاجة لعمل casting للقيمة المستخلصة من ArrayList

```
String s = (String)(list.get(0)); // Casting needed prior to JDK 1.5
```

Autoboxing & Autounboxing

أنواع البيانات المستخدمة عند استعمال generic types يجب أن تكون reference type غير مسموح بأستعمال primitive type، عند الحاجة لاستخدام primitive types يجب أستخدام wrapper classes مثل Integer , Double .

مثال:

```
ArrayList<int> intList = new ArrayList<int>();
```

wrong

```
ArrayList<Integer> intList = new ArrayList<Integer>();
```

OK

مثال:

لانشاء ArrayList لحفظ objects من int نستخدم wrapper class Integer

```
ArrayList<Integer> intList = new ArrayList<Integer>();
```

لحفظ قيمة من نوع int في intList نستخدم التالي :

```
intList.add(5);
```

لغة جافا تقوم بصورة آلية تحويل 5 إلى new Integer(5) وهو ما يعرف ب autoboxing
والذي يعني التحويل من primitive types إلى wrapper class type .

مثال:

عند وضع قيمة من نوع wrapper type في متغير من نوع primitive type تقوم لغة جافا بصورة آلية بتحويلها إلى primitive type وهو مايعرف ب autounboxing.

مثال:

```
1  ArrayList<Double> list = new ArrayList<Double>();
2  list.add(5.5); // 5.5 is automatically converted to new Double(5.5)
3  list.add(3.0); // 3.0 is automatically converted to new Double(3.0)
4  Double doubleObject = list.get(0); // No casting is needed
5  double d = list.get(1); // Automatically converted to double
```

Generic class

هي class المستخدم في الاعلان عنها type parameters .

```
public class Box<T> {
private T t;
    public void set(T t) { this.t = t; }
    public T get() { return t; }
}
```

ولاستخدام class Box يجب أن يمرر نوع البيانات المطلوب أستخدامه بدلا من T كالاتي:

Box<Integer> integerBox = new Box<Integer>();

Generic class

يمكن أيضاً استخدام أكثر من parameter type


```
public interface Pair<K, V> {
    public K getKey();
    public V getValue();
}
```

للاعلان عن generic type يتم أستخدام type parameter بعد أسم class .

مثال:

يستحسن عند استخدام type parameters استخدام أحد الحروف التالية:

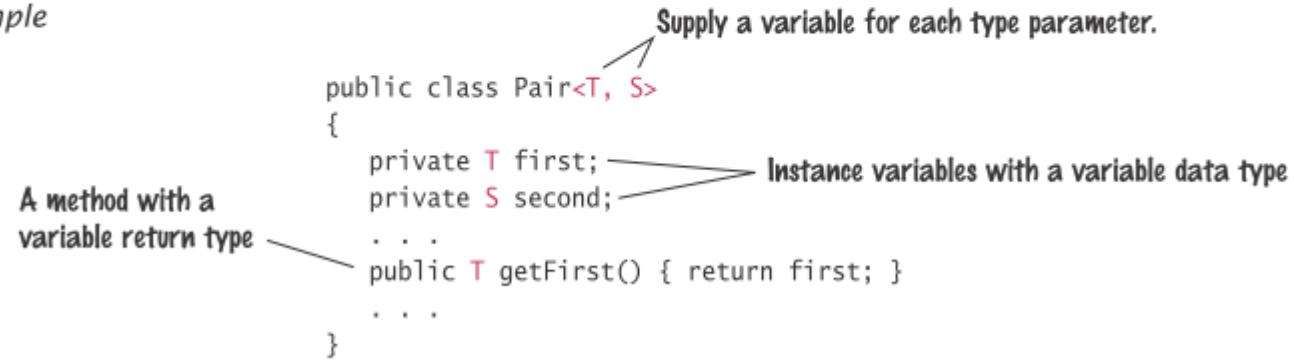
Type Variable	Meaning
E	Element type in a collection
K	Key type in a map
V	Value type in a map
T	General type
S, U	Additional general types

يتم تعريف generic class بالشكل التالي:

Syntax

```
accessSpecifier class GenericClassName<TypeVariable1, TypeVariable2, . . .>
{
   instance variables
   constructors
   methods
}
```

Example



Generic methods

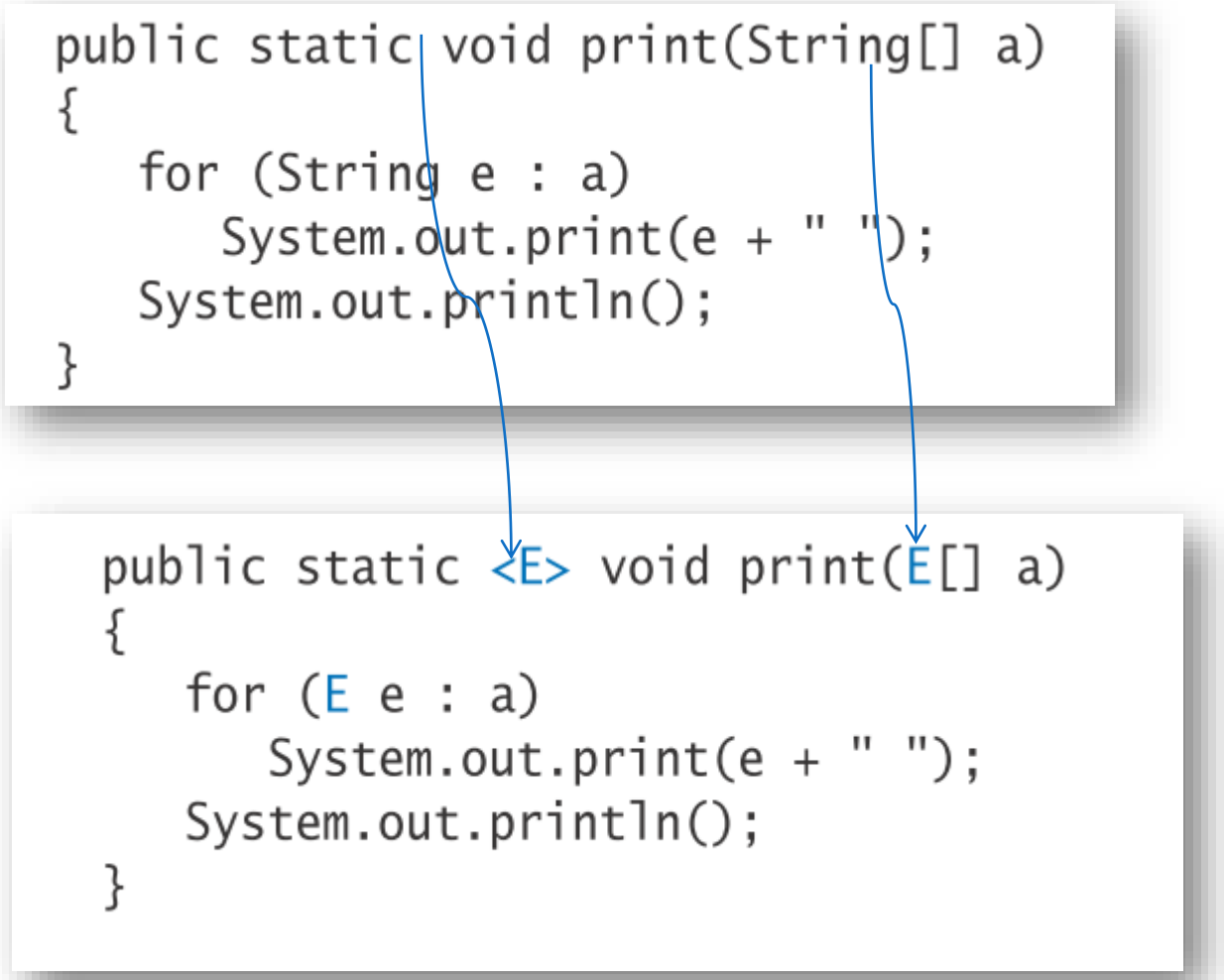
هي عبارة عن methods التي تستخدم type parameters ويكون المجال التي تستخدم محدد فقط ب method المعرفين بها.

Type parameter

```
public class Util {
    public static <K, V> boolean compare(Pair<K, V> p1,
Pair<K, V> p2) {
        return p1.getKey().equals(p2.getKey()) &&
               p1.getValue().equals(p2.getValue());
    }
}
```

للاعلان عن generic method يتم أستخدام type parameter قبل نوع البيانات الذي ترجعه method .

مثال:



```
public static void print(String[] a)
{
   for (String e : a)
      System.out.print(e + " ");
   System.out.println();
}
```

```
public static <E> void print(E[] a)
{
   for (E e : a)
      System.out.print(e + " ");
   System.out.println();
}
```

Bounded Type Parameters

أحياناً نحتاج لتحديد أنواع البيانات التي يمكن استخدامها مكان type parameter ويمكن أن يتم ذلك كالاتي:

Bounded type parameter

```
public class CompareItem<T extends Comparable> {
    private T first;
    private T second;
    public CompareItem(T firstitem, T seconditem) {
        first = firstitem;
        second = seconditem;
    }

    public T max(){
        if (first.compareTo(second)<=0)
            return first;
        else
            return second;
    }

}
```

wildcards (?)

في generic Programming الرمز (?) يعرف ب wildcard ويستخدم للتعبير عن unknown type ويستخدم في عدة حالات هي:

- Upper bounded Wildcards
- Lower bounded Wildcards
- Unbounded Wildcards

Name	Syntax	Meaning
Wildcard with lower bound	`? extends B`	Any subtype of `B`
Wildcard with upper bound	`? super B`	Any supertype of `B`
Unbounded wildcard	`?`	Any type

Unbounded Wildcards

يستخدم عندما نريد أن يستقبل type parameter أي نوع وهي الحالة التي يعرف فيها ب unknown type.

عند الرغبة في كتابة method تعمل على أي نوع يمكن كتابتها كالتالي:

```
printList(ArrayList<?> list)
```

وهو مايعني ان List تستقبل أي نوع.

Lower Bounded Wildcards

يستخدم لتحديد الانواع التي من أن يقبلها type parameter والتي ستكون بين أي type و subtype .

عند الرغبة في كتابة method تعمل على class معينة وكذلك subtypes من تلك class يتم أستخدام upper bound wildcard حسب الصورة التالية:

```
printList( ArrayList<? extends Number>)
```

وهو مايعني ان List تستقبل النوع Number بالاضافة إلى إي subclass ناتجة عنه.

Upper Bounded Wildcards

يستخدم لتحديد الانواع التي من أن يقبلها type parameter والتي ستكون بين أي type و supertype.

عند الرغبة في كتابة method تعمل على class معينة وكذلك supertypes من تلك class يتم أستخدام lower bound wildcard حسب الصورة التالية:

```
printList(ArrayList<? super Integer>)
```

وهو مايعني ان List تستقبل النوع Integer بالاضافة إلى إي superclass لها.

مثال:

```
ArrayList<?> unknownList = new ArrayList<Number>();
unknownList = new ArrayList<Float>();
```

```
ArrayList<? extends Number> numberList = new ArrayList<Number>();
numberList = new ArrayList<Integer>();
numberList = new ArrayList<Float>();
```

```
ArrayList<? super Integer> numberList = new ArrayList<Number>();
numberList = new ArrayList<Integer>();
numberList = new ArrayList<Float>(); //compilation error
```

مثال:

```
public class WildCardExample {
    public static void main(String[] args){
        Integer[] integers = {1, 2, 3, 4, 5};
        ArrayList<Integer> integerList = new ArrayList<Integer>();
      for (Integer element : integers)
          integerList.add(element);
      System.out.printf("integerList contains: %s%n", integerList);
      System.out.printf("Total of the elements in integerList: %.0f%n%n",
        sum( integerList ));

      Double[] doubles = {1.1, 3.3, 5.5};
      ArrayList<Double> doubleList = new ArrayList<Double>();
      for (Double element : doubles)
          doubleList.add(element);
      System.out.printf("doubleList contains: %s%n", doubleList);
      System.out.printf("Total of the elements in doubleList: %.1f%n%n",
          sum(doubleList) );

      Number[] numbers = {1, 2.4, 3, 4.1}; // Integers and Doubles
      ArrayList<Number> numberList = new ArrayList<Number>();
      for (Number element : numbers)
          numberList.add(element);
      System.out.printf("numberList contains: %s%n", numberList);
      System.out.printf("Total of the elements in numberList: %.1f%n",
          sum( numberList));
    } // end main
    public static double sum(  ArrayList<? extends Number>list)
    {
      double total = 0; // initialize total
      for (Number element : list)
          total += element.doubleValue();
      return total;
    }
} // end class WildcardTest
```

شكراً