COMP3021 Lab11

Generics

The World without Generics

- Use non-generic version

  -
    ```
    List nums = new ArrayList();
    nums.add(1);
    nums.add("233");
    ```
- Can we make sure it returns int?
  - `nums.get(0);`
- No type-checking

- When we need compile time type checking
  - `ShortList`
  - `IntList`
  - `LongList`
  - `FloatList`
  - `DoubleList`
  - `ListOfList`
  - `...`

Generics

- Generics enable ***types*** (classes and interfaces) to be parameterized when defining *classes, interfaces* and *methods*.
- A way to re-use the same code with different types of inputs.
- Compile time type checking

Generic Classes and Interfaces

- Defining a generic class

```
class Test<E> {
    public void foo(E obj) { ... }
}
```

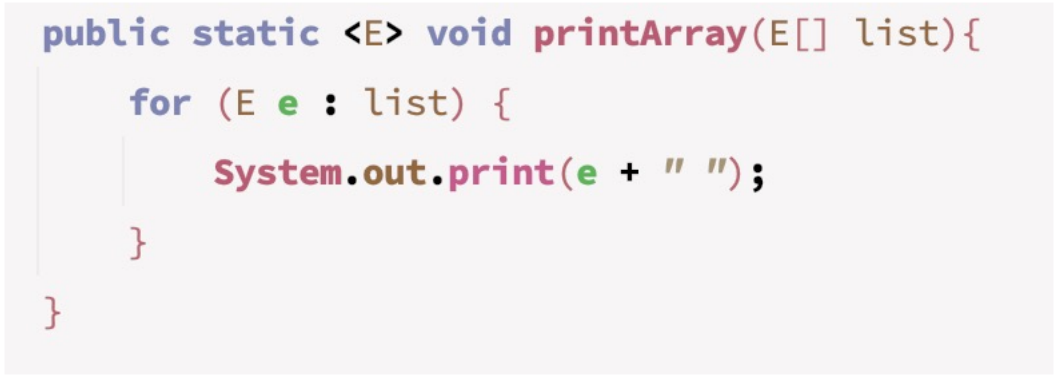
```
interface Comparable<T> {
    public int compareTo(T o);
}
```

- A generic class is shared by all its instances regardless of its actual type.

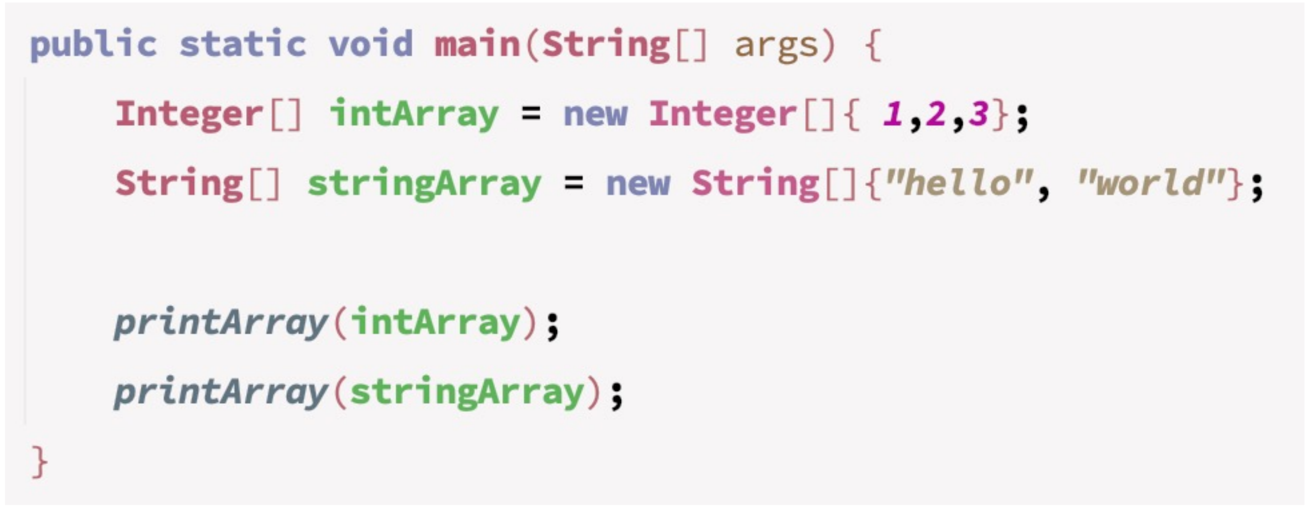
```
Stack<String> s1 = new Stack<>();
Stack<Integer> s2 = new Stack<>();

s1 instanceof Stack; //true
s2 instanceof Stack; //true
```

Generics Method

- Generic methods are written with a single method declaration but can be called with arguments of different types.
  - The type parameter should be placed **before** the return type
  - Generic methods can have more than 1 type parameter, separated by commas in method signature



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



```
public static void main(String[] args) {
    Integer[] intArray = new Integer[]{ 1,2,3};
    String[] stringArray = new String[]{"hello", "world"};

    printArray(intArray);
    printArray(stringArray);
}
```

Generics in Static Context

- Static context should have type parameters of its own

  -
    ```
    public class Test<X> {
      public static void func1(X arg) { } // not allowed
      public static <Y> func2(Y arg) { } // allowed
    }
    ```
- Because the type parameter X belongs to instances of the class

- When trying to understand a limitation of a programming language, try to construct a scenario where there is a conflict

Type Erasing

- Type information is not available at runtime and all the generic stuffs are processed as java.lang.Object.

- Primitive types are not allowed to be type parameters
- Can not make any use of the type parameters at run time
- Exception types can not be generic

Bounded Generics

- We can **restrict** the types that can be accepted by a method.
    - For example, we can specify that we **accept** the type and all its subclasses (using the extends keyword).

```
public <T extends Number> List<T> fromArrayToList(T[] a) { ... }
```

Wildcards and Inheritance

- Is ArrayList<Integer> a subtype of ArrayList<Number> ?
  - No
- Wildcards comes to solve this problem
  - You can read “?” as “anything”.
- Some facts:
  - ArrayList<Object> is a subtype of ArrayList<?>
  - ArrayList<Integer> is a subtype of ArrayList<? extends Number>
  - ArrayList<Number> is a subtype of ArrayList<? extends Number>
  - ArrayList<?> is a subtype of ArrayList<? extends Object>
  - List<? extends Integer> is a subtype of List<? extends Number>

Lab Submission

- Finish the TODOs in Heap.java
- Submit Heap.java to CASS
- Deadline: 23:59 Nov 25