

CS2030/S Programming Methodology

Semester 1 2020/2021

16 September 2020

Problem Set #4 Suggested Guidance

1. Consider a generic class `A<T>` with a type parameter `T` having a constructor with no argument. Which of the following expressions are valid (with no compilation error) ways of creating a new object of type `A`? We still consider the expression as valid if the Java compiler produces a warning.

(a) `new A<int>()`

(b) `new A<>()`

(c) `new A()`

(a) *Error. A generic type cannot be primitive type. Use a wrapper class Integer*

(b) *Valid. Java will create a new class replacing T with Object, but you are advised to be explicit, i.e. new A<Object>() or new A<Integer>().*

(c) *Valid as well. Same behaviour as above, but using raw type (for backwards compatibility) instead. Should be avoided in our class!*

2. In the Java Collections Framework, `List` is an interface that is implemented by both `ArrayList`. For each of the statements below, indicate if it is a valid statement with no compilation error. Explain why.

(a) `void foo(List<?> list) { }`

`foo(new ArrayList<String>())`

(b) `void foo(List<? super Integer> list) { }`

`foo(new List<Object>())`

(c) `void foo(List<? extends Object> list) { }`

`foo(new ArrayList<Object>())`

(d) `void foo(List<? super Integer> list) { }`

`foo(new ArrayList<int>())`

(e) `void foo(List<? super Integer> list) { }`

`foo(new ArrayList());`

- (a) Yes, since `ArrayList<String> <: List<String> <: List<?>`
- (b) No, `List` is an interface. It will be fine if we change it to `ArrayList<Object>` since `ArrayList<Object> <: List<Object> <: List<? super Object> <: List<? super Integer>`
- (c) Yes, since `ArrayList<Object> <: ArrayList<? extends Object> <: List<? extends Object>`
- (d) Error. A generic type cannot be primitive type.
- (e) Compiles, but with a unchecked conversion warning. Use of raw type should also be generally be avoided.

3. In the lecture, we have shown the use of the `Comparator<T>` interface with the abstract method `int compare(T t1, T t2)` that returns zero if `t1` and `t2` are equal, a negative integer if `t1` is less than `t2`, or a positive integer if `t2` is less than `t1`.

A generic method `T max3(T a, T b, T c, Comparator<T> comp)` is defined below. The method takes in three values of type `T` as well as a `Comparator<T>`, and returns the maximum among the values.

```
<T> T max3(T a, T b, T c, Comparator<T> comp) {
    T max = a;
    if (comp.compare(b, max) > 0) {
        max = b;
    }
    if (comp.compare(c, max) > 0) {
        max = c;
    }
    return max;
}
```

- (a) Demonstrate how the `max3` method is called to return the maximum of three integers `-1`, `2` and `-3`.

```
jshell> class IntComparator implements Comparator<Integer> {
...>     public int compare(Integer i1, Integer i2) {
...>         return i1 - i2;
...>     }
...> }
jshell> max3(-1, 2, -3, new IntComparator());
$.. ==> 2
```

- (b) Other than `Comparator<T>`, there is a similar `Comparable<T>` interface with the abstract method `int compareTo(T o)`. This allows one `Comparable` object to compare itself against another `Comparable` object. Now we would like to redefine the `max3` method to make use of the `Comparable` interface instead.

```
<T> T max3(T a, T b, T c) {
    T max = a;
```

```

    if (b.compareTo(max) > 0) {
        max = b;
    }
    if (c.compareTo(max) > 0) {
        max = c;
    }
    return max;
}

```

Does the above method work? What is the compilation error?

```

jshell> /open ...
| Error:
| cannot find symbol
|   symbol:   method compareTo(T)
|   if (b.compareTo(max) > 0) {
|       ^-----^

```

There is no guarantee that an object of type T implements the Comparable<T> interface

(c) Now, we further restrict T to be Comparable<T>

```

<T extends Comparable<T>> T max3(T a, T b, T c) {
    T max = a;
    if (b.compareTo(max) > 0) {
        max = b;
    }
    if (c.compareTo(max) > 0) {
        max = c;
    }
    return max;
}

```

Demonstrate how the method `max3` can be used to find the maximum of three values `-1`, `2` and `-3`. Explain how it works now.

According to the Java API Specification, the `Integer` class implements `Comparable<Integer>` and hence the `compareTo` method is implemented.

```

jshell> max3(-1, 2, -3)
$.. ==> 2

```

(d) What happens if we replace the method header with each of the following:

- i. `<T> Comparable<T> max3(Comparable<T> a, Comparable<T> b, Comparable<T> c)`
Realize that now the method returns a `Comparable<T>` object. If we change the type of `max` to `Comparable<T>`, then we are required to typecast in the argument of the `compareTo` method as it expects an argument of type `T`, e.g. `b.compareTo((T) max)`

```
jshell> @SuppressWarnings("unchecked")
...> <T> Comparable<T> max3(Comparable<T> a, Comparable<T> b, Comparable<T> c) {
...>     Comparable<T> max = a;
...>     if (b.compareTo((T) max) > 0) {
...>         max = b;
...>     }
...>     if (c.compareTo((T) max) > 0) {
...>         max = c;
...>     }
...>     return max;
...> }
jshell> max3(-1, 2, -3)
$2 ==> 2
jshell>
```

- ii. `<T> T max3 (Comparable<T> a, Comparable<T> b, Comparable<T> c)`
The above preserves the return type as `T`. As `a`, `b` and `c` has a type of `Comparable<T>`, there is a type mismatch. An explicit typecasting is therefore required when assigning, say `b` to `max`, e.g. `max = (T) b;`

```
jshell> @SuppressWarnings("unchecked")
...> <T> T max3(Comparable<T> a, Comparable<T> b, Comparable<T> c) {
...>     T max = (T) a;
...>     if (b.compareTo(max) > 0) {
...>         max = (T) b;
...>     }
...>     if (c.compareTo(max) > 0) {
...>         max = (T) c;
...>     }
...>     return max;
...> }
jshell> max3(-1, 2, -3)
$2 ==> 2
jshell>
```

- iii. `Comparable max3(Comparable a, Comparable b, Comparable c)`
As `Comparable` is a generic interface, by not passing any type argument we have created a raw type. Indeed, this code fragment shows the effect of type erasure. When the compiler replaces the type-parameter information with the bound in the method declaration, it also inserts explicit cast operations in front

of each method call to ensure that the returned value is of the type expected by the caller. For example,

```
jshell> @SuppressWarnings("unchecked")
...> Comparable max3(Comparable a, Comparable b, Comparable c) {
...>     Comparable max = a;
...>     if (b.compareTo(max) > 0) {
...>         max = b;
...>     }
...>     if (c.compareTo(max) > 0) {
...>         max = c;
...>     }
...>     return max;
...> }
jshell> max3(-1, 2, -3)
$2 ==> 2
jshell> (Integer) max3(-1, 2, -3)
$3 ==> 2
jshell> /var
|     Comparable $2 = 2
|     Integer $3 = 2
jshell>
```