CS2030 Programming Methodology
Semester 2 2018/2019

6 March – 8 March 2019
Tutorial 4 Suggested Guidance
Generics and Collections

1. For each of the statements below, indicate if it is a valid statement with no compilation error. Explain why.

   (a) `List<?> list = new ArrayList<String>();`

   (b) `List<? super Integer> list = new List<Object>();`

   (c) `List<? extends Object> list = new LinkedList<Object>();`

   (d) `List<? super Integer> list = new LinkedList<>();`

   (a) Yes. `ArrayList` implements `List` and the wildcard type is bound to `String`.

   (b) No. Cannot instantiate an interface `List`.

   (c) Yes. `LinkedList` implements `List` and `Object` is upperbounded by `? extends Object`.

   (d) Yes. `LinkedList` implements `List` and Java infers the type to be `Integer`.

2. 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.

   (b) Ok. Java will create a new class replacing `T` with Object.

   (c) Ok too. Same behavious as above, but using raw type (for backward compatibility) instead. Should be avoided in our class.

3. Given the following Java program fragment,

```
class Main {
    public static void main(String[] args) {
        double sum = 0.0;

        for (int i = 0; i < Integer.MAX_VALUE; i++) {
            sum += i;
        }
    }
}
```

you can determine how long it takes to run the program using the `time` utility

```
$time java Main
```

Now, replace `double` with the wrapper class `Double` instead. Determine how long it takes to run the program now. What inferences can you make?

Despite it's conveniences, there is an associated overhead in the use of autoboxing. In addition, due to immutability of `Integer`, many objects are created.

4. Recall that the `==` operator compares only references, i.e. whether the two references are pointing to the same object. On the other hand, the `equals` method is more flexible in that it can override the method specified in the `Object` class.

In particular, for the `Integer` class, the `equals` method has been overridden to compare if the corresponding `int` values are the same or otherwise.

What do you think is the outcome of the following program fragment?

```
Integer x = 1;
Integer y = 1;
System.out.println(x == y);

x = 1000;
y = 1000;
System.out.println(x == y);
```

Why do you think this happens? *Hint: check out Integer caching*

We would expect the top fragment to be false since we are comparing object references. Since integers within a small range ar very often used, it makes sense for the `Integer` class to keeps a cache of Integer objects within this range (-128 to 127) such that autoboxing, literals and uses of Integer.valueOf() will return instances from that cache instead.

Rather than worry over the effects of caching or otherwise, the bottomline is to always use equals to compare two reference variables.

5. Compile and run the following program fragments and explain your observations.

(a)
```
import java.util.List;

class A {
    void foo(List<Integer> integerList) {}
    void foo(List<String> StringList) {}
}
```

(b)
```
class B<T> {
    T x;
    static T y;
}
```

(c)
```
class C<T> {
    static int b = 0;
    T y;

    C() {
        this.b++;
    }

    public static void main(String[] args) {
        C<Integer> x = new C<>();
        C<String> y = new C<>();

        System.out.println(x.b);
        System.out.println(y.b);
    }
}
```

(a) Overloaded

```
class A {
    void foo(List<Object> integerList) {}
    void foo(List<Object> StringList) {}
}
```

(b) There is only one class `B`. For the field declaration `T x`, the type of `X` is bounded to the type argument `T`, this is fine for instance fields. However for class fields, there is only one copy of `y`. Which type argument should it be bounded to?

(c)
```
2
2
```
Although it seems there are two different classes, `C<Integer>` and `C<String>`, there is still only one class `C`. As such, there is only one copy of the class variable `b`.

6. Which of the following code fragments will compile? If so, what is printed?

(a)
```
List<Integer> list = new ArrayList<>();
int one = 1;
Integer two = 2;

list.add(one);
list.add(two);
list.add(3);

for (Integer num : list) {
    System.out.println(num);
}
```

(b)
```
List<Integer> list = new ArrayList<>();
int one = 1;
Integer two = 2;

list.add(one);
list.add(two);
list.add(3);

for (int num : list) {
    System.out.println(num);
}
```

(c)
```
List<Integer> list = Arrays.asList(1, 2, 3);

for (Double num : list) {
      System.out.println(num);
}
```

(d)
```
List<Integer> list = Arrays.asList(1, 2, 3);

for (double num : list) {
    System.out.println(num);
}
```

(e)
```
List<Integer> list = new LinkedList<>();
list.add(5);
list.add(4);
list.add(3);
list.add(2);
list.add(1);

Iterator<Integer> it = list.iterator();
while (it.hasNext()) {
    System.out.println(it.next());
}
```

(a)
```
1
2
3
```

(b)
```
1
2
3
```

(c)
```
prog.java:8: error: incompatible types: Integer cannot be converted to Double
for (Double num : list) {
                  ^
1 error
```

(d)
```
1.0
2.0
3.0
```

(e)
```
5
4
3
2
1
```