CS2030 Programming Methodology

Semester 1 2023/2024

20 & 21 September 2023

Problem Set #4 Suggested Guidance

Java Generics

1. Given the following declaration of variable `x`,

   ```
   jshell> int x = 1
   x ==> 1
   ```

   which of the following statements will result in a compilation error?

   (a) `Integer i = x`
   (b) `int x = i`
   (c) `double y = x`
   (d) `int x = y`
   (e) `Double d = y`
   (f) `double y = d`
   (g) `Integer i = d`
   (h) `Double d = i`
   (i) `Integer i = y`
   (j) `Double d = x`
   (k) `Double y = i`

   ```
   jshell> int x = 1
   x ==> 1

   jshell> Integer i = x // boxing
   i ==> 1

   jshell> x = i // unboxing
   x ==> 1

   jshell> double y = x // int <: double
   y ==> 1.0

   jshell> x = y // double <: int does not hold
   |  Error:
   |  incompatible types: possible lossy conversion from double to int
   |  x = y // double <: int does not hold
   |      ^

   jshell> Double d = y // boxing
   d ==> 1.0

   jshell> y = d // unboxing
   y ==> 1.0

   jshell> i = d // Integer <: Double does not hold
   |  Error:
   |  incompatible types: java.lang.Double cannot be converted to
   java.lang.Integer
   |  i = d // Integer <: Double does not hold
   |      ^
   ```

```
jshell> d = i // Double <: Integer does not hold
|  Error:
|  incompatible types: java.lang.Integer cannot be converted to
java.lang.Double
|  d = i // Double <: Integer does not hold
|      ^

jshell> i = y
|  Error:
|  incompatible types: double cannot be converted to
java.lang.Integer
|  i = y
|      ^

jshell> d = x
|  Error:
|  incompatible types: int cannot be converted to java.lang.Double
|  d = x
|      ^

jshell> y = i // Unboxing i to int, then int <: double
y ==> 1.0
```

2. For each of the code fragments below, indicate and explain the source of the error(s).

   (a) `List<? extends Object> list = new ArrayList<Object>()`

   > `list` *assignment is valid since (read* `<:` *as "is substitutable for")...*
   > `ArrayList<Object> <: ArrayList<? extends Object> <: List<? extends Object>`

   `list.add(new Object())`

   > `list.add(new Object())` *is invalid since* `list` *could refer to* `ArrayList<Integer>`

   (b) `List<? extends Object> list = List.of("abc");`

   > `list` *assignment is valid since*
   > `List<String> <: List<? extends String> <: List<? extends Object>`

   ```
   list.add("def");
   String s = list.get(0);
   ```

   > *Both* `list.add("abc")` *and* `String s = list.get(0)` *aree invalid since* `list` *could refer to* `ArrayList<Integer>`*. However,* `Object o = list.get(0)` *is fine.*

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

   > `list` *assignment is invalid since* `List` *is an interface. It wil be fine if we change* `List<Object>` *to* `ArrayList<Object>` *since*
   > `ArrayList<Object> <: List<Object> <: List<? super Object> <: List<? super Integer>`

`list.add(new Object())`

`list.add(new Object())` *is invalid since* `list` *could refer to* `ArrayList<Integer>`. *However,* `list.add(1)` *is fine;* `Integer i = list.get(0)` *is invalid; only* `Object obj = list.get(0)` *is valid, though not useful.*

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

Error. A generic type cannot be primitive type.

(e) `List<? super Integer> list = new ArrayList();`

Compiles, but with a unchecked conversion warning. Use of raw type should be avoided.

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

`List<?>` *can refer to all lists!* `list` *assignment is valid since* `ArrayList<String> <: List<String> <: List<?>`.

`list.add("abc");`

`list.add("abc")` *is invalid since* `list` *could refer to* `ArrayList<Integer>`.