CS2030 Programming Methodology

Semester 2 2019/2020

13 February 2020

Problem Set #4

1. Don't forget your diamond <>

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.

   type argument of a generic class has to be a reference type, but int is a primitive type => error

   (a) `new A<int>()` invalid: int -> Integer

   (b) `new A<>()` valid — if you don't put in any type, java will try to infer the type for you. Ex: A<Integer> a = new A<>()

   (c) `new A()` valid with warning — shouldn't use in practice

   for ex, for the class A in the note, if we call A a = new A() (this is called raw type) then we run a.put("abc"), the program cannot check whether "String" is of valid type or not. "abc" will be treated as an object.
   -If we call String s = a.get() it will return an error because the program returns "abc" as an object, not a string. this error can be dismiss if you run String s = (String) a.get() instead
   - If we run Integer s = (Integer) a.get()
   => no compilation error, but there is runtime error
   Conclusion: if use raw type, ignoring the warning, could get runtime error. If not use raw type, will return compile time error => easier to fix

2. 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;
        }
    }
}
```

2. Don't use wrapper class for primitive types unnecessarily

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

```
$time java Main
```

   need more time to "unbox" sum Double to double and auto "box" double to Double again
   wrapper objects are immutable => every time add i, we have to create a new object

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

3. 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;
x == y
```

   Integer x = 1 <=> Integer x = new Integer(1)

   true

   Integer caching: From java 5, there is an array that store Integer object from -127 to 128 in order to excessive creation of objects

```
x = 1000;
y = 1000;
x == y
```

   false

   3. Always .equals() to compare to objects

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

4. In the Java Collections Framework, `List` is an interface that is implemented by `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>());
   ```
   valid

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

   foo(new List<Object>());
   ```
   invalid: can never create an object of interface - type

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

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

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

   foo(new ArrayList<int>());
   ```
   invalid: type argument must be reference type

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

   foo(new ArrayList());
   ```
   valid with a warning of not type - safe