Welcome to CS2030S Lab 5!

8 October 2021 [16A]

Please login to your pe node once it's 4pm.

PA1

- Please submit your final changes by **Sunday 2359**.
- A sample solution will be uploaded onto LumiNUS for your reference **after** the submission deadline.

PA2

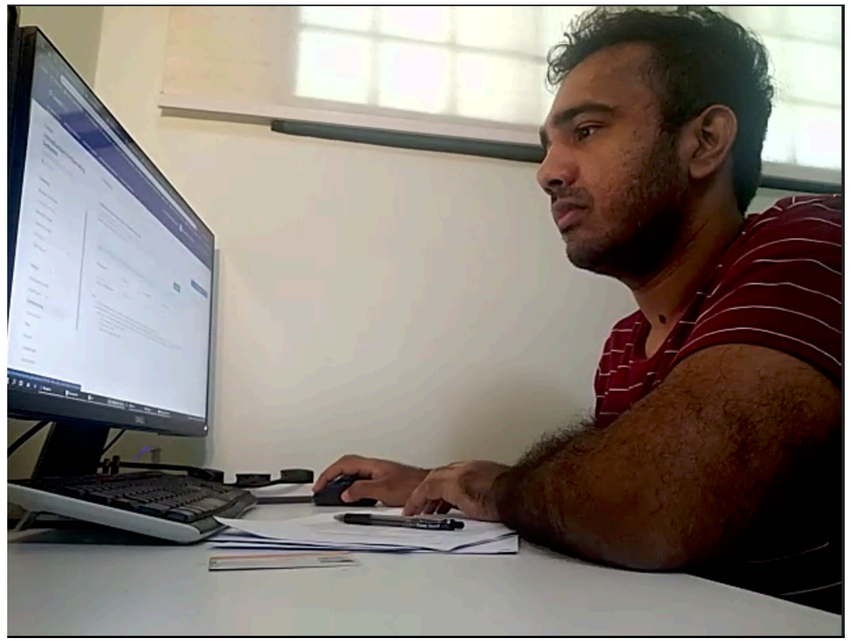
- In lieu of the current COVID-19 situation, PA2 may be conducted online.
- We will need everyone to test their screen recording software to ensure that it’s working well.

PA2 Admin - Screen Recording

- We recommend using FFMPEG
- You may find instructions on installing and using FFMPEG here: https://mysoc.nus.edu.sg/academic/e-exam-sop-for-students/
- You may also use any other screen recording of your choice
  Please upload a sample recording onto LumiNUS after your lab for us to verify that your screen recording software is working as intendedYou should upload the recording under Multimedia -> Lab 5 (<Your lab group>)

PA2 Admin Camera Setup

Please also ensure that you are able to set up your camera as shown below for proctoring purposes



Submission Statistics

Labs	Submitted	Not Submitted	A
1🟢	17	0	17🤩
2🟡	17	0	16
3🟡	16	1	12
4🟡	17	0	15
PA1🔴	17	0	8
Project😂	1	16	0

Mini Lecture on Generics

- You have used generic class `PriorityQueue` and generic interfaces `Comparable` and `Comparator` in Lab 4!
- *Generics* let you parameterise types - define a class or method with generic types that the compiler can replace with concrete types.
- e.g.: From the generic class `PriorityQueue`, you can create `PriorityQueue` objects for holding strings or numbers. Here strings and numbers are concrete types that replace the generic type.

- A generic class or method permits you to specify allowable types of objects that the class or method can work with. **If you attempt to use an incompatible object, the compiler will detect that error**.
- By convention, a single capital letter such as `E` or `T` is used to denote a formal generic type.
- Generic types must be reference types. You cannot replace a generic type with a primitive type such as `int`, `double`, or `char`. For example the following statement is wrong

```
ArrayList<int> intList new ArrayList<>();
```

Defining Generic Classes and Interfaces

A generic type can be defined for a class or interface. A concrete type must be specified when using the class to create an object or using the class or interface to declare a reference variable.

- Occasionally, a generic class may have more than one parameter. In this case, place the parameters together inside the brackets, separated by commas—for example, `<E1, E2, E3>`.

Generic Methods

A generic type can be defined for a static method.

```
public class GenericMethodDemo {
    public static void main(String[] args) {
        Integer[] integers = {1, 2, 3, 4, 5};
        String[] stirngs = {"London", "Paris", "New York", "Austin"};

        // invoking generic method
        GenericMethodDemo.<Integer>print(integers);
        GenericMethodDemo.<String>print(strings);
    }

    public static <E> void print(E[] list) { // declaring generic method
        for (int i = 0; i < list.length; i++)
            System.out.print(list[i] + " ");
        System.out.println();
    }
}
```

Generic Methods

A generic type can be defined for a static method.

```
public class GenericMethodDemo {
    public static void main(String[] args) {
        Integer[] integers = {1, 2, 3, 4, 5};
        String[] stirngs = {"London", "Paris", "New York", "Austin"};

        // also works, compiler automatically discovers the actual type.
        print(integers);
        print(strings);
    }

    public static <E> void print(E[] list) { // declaring generic method
        for (int i = 0; i < list.length; i++)
            System.out.print(list[i] + " ");
        System.out.println();
    }
}
```

Bounds for Type Variables

We want to compute the smallest element of an array. **What's wrong?**

```
public static <T> T min(T[] a) {
    if (a == null || a.length == 0)
        return null;
    T smallest = a[0];
    for (int i = 1; i < a.length; i++)
        if (smallest.compareTo(a[i]) > 0)
            smallest = a[i];
    return smallest
}
```

How do we know that the class to which T belongs has a `compareTo` method?

Solution

```
public static <T extends Comparable<T>> T min(T[] a)...
```

1. T is a subtype of `Comparable`.

2. The elements to be compared are of the T type.

Wildcard Types

You can use **unbounded wildcards**, **bounded wildcards**, or **lower bound wildcards** to specify a range for a generic type.

1. `?`: unbounded wildcard, is the same as `? extends Object`.
2. `? extends T`: (upper) bounded wildcard, represents `T` or a **subtype** of `T`.
3. `? super T`: lower bound wildcard, denotes `T` or a **supertype** of `T`.

Note that `GenericClass<Integer>` is not a subtype of `GenericClass<Object>` even though `Integer` is a subtype of `Object`.

PECS: Producer-`extends`, consumer-`super`

Get and Put Principle

- Use an **extends** wildcard when you only **get** values out of a structure.
- Use a **super** wildcard when you only **put** values into a structure.
- And don't use a wildcard when you both get and put.

2 Exceptions:

1. You cannot put anything into a type declared with an extends wildcard except for the value `null`, which **belongs to every reference type**.
2. You cannot get anything out from a type declared with a super wildcard except for a value of type `Object`, which is a **super type of every reference type**.

Java Generics

```
public class Box<T> {
    private T item;
    public Box(T item) {
        this.item = item;
    }
}
```

T is a generic type.

Declaring `Box<Integer>` replaces all instances of `T` in your code with `Integer`.

Java Generics

- Every instance of `Box` can have a different type that is assigned to `T`.
- Therefore, `T` belongs to an **instance** of `Box`.
- How do we insert a generic type into a static method or variable?

Java Generics

```
public class Box<T> {
    private T item;
    public static <T> Box<T> empty() {
        // Pink T replaces all the Ts in this method
    }
}
```

Solution: Declare <T> in front of a static method.

Java Generics

⚠️ The T's on the right side **only exist within the scope of the static method**, while the T's on the left exist in all other **instance attributes** of `Box`.

```
public class Box<T> {
    private T item;
    public static <T> Box<T> empty() {
        // code
    }
}
```

```
public class Box<T> {
    private T item;
    public static <T> Box<T> empty() {
        // code
    }
}
```

Java Generics

⚠️ The T's on the right side **only exist within the scope of the static method**, while the T's on the left exist in all other **instance attributes** of `Box`.

```
public class Box<T> {
    private T item;
    public static <T> Box<T> empty() {
        // code
    }
}
```

```
public class Box<T> {
    private T item;
    public static <T> Box<T> empty() {
        // code
    }
}
```

Map

- A map is a data structure which maps a key to a value.
- One key can only be mapped to one value.
- Mapping a an existing key to another value will replace the current mapping in the map.
- Think of it as a dictionary (for students familiar with Python/ JavaScript).

`java.util.HashMap<K, V>`

- HashMaps are Maps which are backed by a hash table (you will learn more about this in CS2040/C/S).
- They require the use of two generic types (one for the key and one for the value).
- To declare a HashMap with keys of type A and values of type B:

```
HashMap<A, B> map = new HashMap<A, B>();
```

java.util.HashMap<K, V>

Update Methods	Function
`clear(): void`	Remove all entries from this map
`put(key: K, value V): V`	Puts an entry into this map.
`putAll(m: Map<? extends K, ? extends V>): void`	Adds all the entries from `m` to this map.
`remove(key: Object): V`	Removes the entry for the specified key.

java.util.HashMap<K, V>

Query Methods	Function
`containsKey(key: Object): boolean`	Returns true if this map contains an entry for the specified key.
`containsValue(value: Object): boolean`	Returns true if this map maps one or more keys to the specified value.
`isEmpty(): boolean`	Returns true if this map contains no entries.
`size(): int`	Returns the number of entries in this map.

java.util.HashMap<K, V>

Other Methods	Function
`keySet(): Set<K>`	Returns a set consisting of the keys in this map.
`entrySet(): Set<Map.Entry<K, V>>`	Returns a set consisting of the entries in this map. The entries instances of the `Map.Entry<K, V>` interface below.