CSE 12 SP24 Exam 1

Do not begin until instructed.

This exam is closed book, closed notes. No study aids or electronic devices are allowed. Turn off your phone and have no bags open.

You must hand in all sheets, including scratch paper, at the end of the exam.

This exam is designed to assess **your** understanding of topics from CSE12. You cannot communicate with other students during the exam. You cannot discuss this exam with anyone from now until you receive your grade on the exam. This includes posting about the exam on Piazza or elsewhere online.

Please write “I excel with integrity” in the box below:

Sign your name: _____

Print your name: _____

Print your email: _____

PID: _____

You have 45 minutes to complete the exam. Work to maximize points. **Write all your answers in the provided answer sheet on the back of this page**. There are **23 points** total. If you get stuck, work through other problems, and come back to it.

Once the exam starts, you can rip off/remove the answer sheet.

In general, if you think you've spotted a typo in the exam, do your best to answer in the spirit of the question. Keep in mind that some questions have interesting code examples with intentional bugs for you to find as part of the question. If you ask a question during the exam, we may decline to answer. In general, assume that any necessary libraries (JUnit, ArrayList, List, and so on) have been imported.

Stay calm – you can do this!

Do not begin until instructed.

Question 1 3 points each for implementation, 1 point each for C/D

A: implementation of maybeEnqueue()

B: implementation of dequeue()

C	D

Question 2 (1 point each)

A	B

Question 3 (5 points for A, 2 points for B)

A: implementation of replaceValue()

B: implementation of testStack()

Question 4 (1 point each)

A	B	C	D

Question 5 (1 point)

Question 6 (1 point)

Reference

Module java.base

Package java.util

Class ArrayList<E>

Type Parameters:

`E` - the type of elements in this list

All Implemented Interfaces:

`Serializable`, `Cloneable`, `Iterable<E>`, `Collection<E>`, `List<E>`, `RandomAccess`

Module java.base

Package java.util

Interface List<E>

`void`	**`add`**`(int index, E element)`	Inserts the specified element at the specified position in this list.
`boolean`	**`add`**`(E e)`	Appends the specified element to the end of this list.
`E`	**`get`**`(int index)`	Returns the element at the specified position in this list.
`int`	**`indexOf`**`(Object o)`	Returns the index of the first occurrence of the specified element in this list, or -1 if this list does not contain the element.
`E`	**`remove`**`(int index)`	Removes the element at the specified position in this list.
`boolean`	**`remove`**`(Object o)`	Removes the first occurrence of the specified element from this list, if it is present.
`int`	**`size`**`()`	Returns the number of elements in this list.

Question 1

Consider implementing the interface `BoundedNumberQueue`:

```
interface BoundedNumberQueue<E> {
  int size();
  int getMaxNumber();
  void maybeEnqueue(E element, int number);
  E dequeue();
}
```

A `BoundedNumberQueue` is like a regular `Queue`, but each element has an associated number when being enqueued to the Queue. Only the elements with an associated number **lower** than the maximum number can be enqueued (by `maybeEnqueue`). Dequeue for `BoundedNumberQueue` should dequeue the elements like a regular `Queue`, and it should **explicitly** throw some exception if the queue is empty.

```
class ALBNQueue<E> implements BoundedNumberQueue<E> {
  ArrayList<E> contents;
  final int maxNumber;        // the fixed maximum number the queue accepts

  public ALBNQueue(int maxNumber) {
    this.contents = new ArrayList<E>();
    this.maxNumber = maxNumber;
  }

  public int size() { return this.contents.size(); }

  public int getMaxNumber() { return this.maxNumber; }

  public void maybeEnqueue(E element, int number) {
    // make sure to answer 1A directly in the answer sheet
  }

  public E dequeue() {
    // make sure to answer 1B directly in the answer sheet
  }
}
```

A. Provide an implementation of **`maybeEnqueue`** that matches the description above.

B. Provide an implementation of **`dequeue`** that matches the description above.

C. **True or false (Do <u>not</u> write T/F, write the complete word):** If we change the interface `BoundedNumberQueue` to the version below, we would need to edit `ALBNQueue` to fix compile errors:

```
interface BoundedNumberQueue<Element> {
  int size();
  int getMaxNumber();
  void maybeEnqueue(Element element, int number);
  Element dequeue();
}
```

D. **True or false (Do <u>not</u> write T/F, write the complete word):** Changing the type of the contents field from `ArrayList<E>` to `List<E>` would cause a compile error.

Question 2

A. Consider this test case:

```
@Test
public void testStack() {
  List m = new MyStack();
  m.add(7);
  m.add(8);
  m.remove();
  m.add(6);
  m.add(2);
  m.remove();
  m.remove();
  int x = m.remove();
  // make sure to answer 2A directly in the answer sheet
}
```

Given that MyStack is a Stack, write an **assertEquals** test to check that the letter that was removed and stored in variable **x** is the expected letter.

B. Consider this test case:

```
@Test
public void testQueue() {
  List m = new MyQueue();
  m.add(7);
  m.add(8);
  m.add(6);
  m.remove();
  m.remove();
  m.add(3);
  m.remove();
  int y = m.remove();
  // make sure to answer 2B directly in the answer sheet
}
```

Given that MyQueue is a Queue, write an **assertEquals** test to check that the letter that was removed and stored in variable **y** is the expected letter.

Question 3

```
class Node<E> {
  E value;
  Node<E> next;
  public Node(E value, Node<E> next) {
    this.value = value;
    this.next = next;
  }
}
```

A `MusicStack` is similar to a standard stack. However, it introduces the functionality of `replaceValue()`, which finds the element with the specified value in the stack and replaces it with the new value. This operation is performed in addition to the typical stack operations such as `push()`, `pop()`, and `peek()`. Assume that all elements in the stack are unique. Note that the LinkedList will have a dummy node `top` as used in the PAs.

```
public class MusicStack<E> {
  Node<E> top;               // dummy node
  int size;

  public MusicStack() {
    this.top = new Node<E>(null, null);
    this.size = 0;
  }

  /**
  * Finds the element with the value val, and replaces it with the new value newVal
  * Throws an exception if the element with value val does not exist in the stack.
  */
  void replaceValue(E val, E newVal) throws IllegalArgumentException {
    // make sure to answer 3A directly in the answer sheet
  }

  // Pushes element elt onto the top of the stack
  void push(E elt) { // Assume implemented correctly }

  // Pops the top element from the stack and returns it
  E pop() { // Assume implemented correctly }

  // Peeks at the top element of the stack without removing it
  E peek() { // Assume implemented correctly }

  @Test
  public void testStack() {
    MusicStack<Integer> testStack = new MusicStack<>();
    // make sure to answer 3B directly in the answer sheet
  }
}
```

A. Provide an implementation of `replaceValue()` that matches the description above.

B. Write a test with a simple case to show that `replaceValue()` correctly replaces the value of an element in a stack. This test case **shouldn't** invoke an exception.

Question 4

Consider this class and interface hierarchy:

```
interface MyInterface {
    int calculate(int x);
    void print(String message);
}

class MyClass1 implements MyInterface {
    /* questions about this class body below */
}

class MyClass2 implements MyInterface {
    /* questions about this class body below */
}
```

A. **True or false (do <u>not</u> write T/F, write the complete word)**: We can have a method in the body of `MyClass1` which has the following header:
`void calculate(String input) {...}`

B. **True or false (do <u>not</u> write T/F, write the complete word)**: Given that the `MyClass1` and `MyClass2` bodies compile, this line compiles:
`MyClass1 obj = new MyInterface();`

C. Given that the body of `MyClass1` compiles, and `MyClass1` has the following two methods with the headers:
`void print(int value) {...}`
`void process(String input) {...}`
`boolean isValidInput(String input) {...}`
`MyClass1` must implement at least __ total methods. **Write a number.**

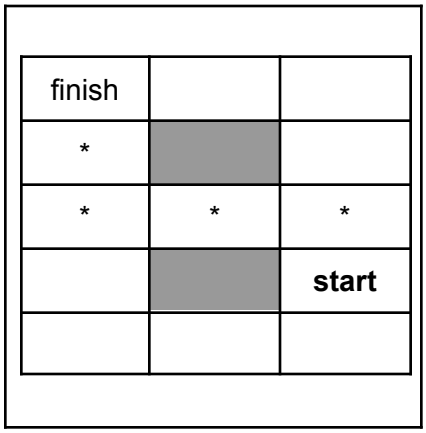
D.Suppose we added the following method to `MyClass1`'s and `MyClass2`'s body:
`void update(int value) {...}`

And we create the following objects:
```
MyInterface obj = new MyClass1();
MyClass2 obj2 = new MyClass2();
```

Which of the following lines of code will encounter a compilation error?
A. `obj2.update(5);`
B. `obj2.print("hello");`
C. `obj.update(5);`
D. `obj.print(“hello”);`

Question 5

The maze below was solved with a **stack worklist/DFS**. The asterisks mark the path from the finish back to the start by following previous references.



For the above case, various **orders** have been chosen for adding the available neighbors to the worklist. You will answer which **order** matches with the above maze solution. **Choose from the answers below**. A reminder that on the page, **West** is left, **East** is right, **North** is up, and **South** is down.

- **A.** **ESNW** for East then South then North then West
- **B.** **NSEW** for North then South then East then West
- **C.** **WSEN** for West then South then East then North
- **D.** **NWSE** for North then West then South then East

```
initialize wl to be a new empty worklist (stack _or_ queue)
add the start square to wl
mark the start as visited
while wl is not empty:
  let current = remove the first element from wl (pop or dequeue)
  if current is the finish square
    return current
  else
    for each neighbor of current that isn't a wall and isn't visited IN SOME ORDER
      mark the neighbor as visited
      set the previous of the neighbor to current
      add the neighbor to the worklist (push or enqueue)

if the loop ended, return null (no path found)
```

Question 6

For the maze in question 5, could it have been the solution from a **queue worklist/BFS** for **any** of the given orderings? Write **Yes** or **No** directly in the answer sheet. No further solution needs to be provided.

Scratch Paper