INFO1113 / COMP9003
Object-Oriented Programming

Lecture 13

These slides will be released after this lecture

THE UNIVERSITY OF SYDNEY

Acknowledgement of Country

I would like to acknowledge the Traditional Owners of Australia and recognise their continuing connection to land, water and culture. I am currently on the land of the Gadigal People of the Eora nation and pay my respects to their Elders, past, present and emerging.

I further acknowledge the Traditional Owners of the country on which you are on and pay respects to their Elders, past, present and future.

Copyright Warning

COMMONWEALTH OF AUSTRALIA

Copyright Regulations 1969

WARNING

This material has been reproduced and communicated to you by or on behalf of the University of Sydney pursuant to Part VB of the Copyright Act 1968 (**the Act**).

The material in this communication may be subject to copyright under the Act. Any further copying or communication of this material by you may be the subject of copyright protection under the Act.

Do not remove this notice.

Topics

- **Exam format (covered in week 12)**
- **Review contents**

Final Exam

- Date: 15 Nov 2024
- Time: 13:00 Sydney time
- Duration: 130 Minutes
  - Reading time: 10 Minutes
  - Writing time: 120 Minutes
- Exam adjustment is done by the **exam office**
  - Notification no later than 3 days before the exam

Remember the Double Pass Criteria

To get a pass, you must get ≥ 40% in the final exam AND your total marks must be ≥ 50%

In-semester Mark 44%, Final Exam Mark 50%, total 47%: Fail

In-semester Mark 75%, Final Exam Mark 35%, total 55%: Fail

THE UNIVERSITY OF SYDNEY

Room Number	
Seat Number	
Student Number	

ANONYMOUSLY MARKED
(Please do not write your name on this exam paper)

CONFIDENTIAL EXAM PAPER

This paper is not to be removed from the exam venue

Computer Science

EXAMINATION

Semester 2 - Final, 2024

INFO1113 Object-Oriented Programming

EXAM WRITING TIME: 2 hours

READING TIME: 10 minutes

EXAM CONDITIONS:
This is a CLOSED book exam - no material permitted

MATERIALS PERMITTED IN THE EXAM VENUE:
(No electronic aids are permitted e.g. laptops, phones)

Calculator - non-programmable

MATERIALS TO BE SUPPLIED TO STUDENTS:
Answer sheet: Gradescope MCQ (single-sided - 100 Qs)

INSTRUCTIONS TO STUDENTS:

- This exam consists of three parts.
- Part A and B contain 10 Multiple-Choice Questions (MCQ) worth 14 marks.
- Writing on this MCQ sheet will not be considered for marking. Your response to the MCQs should be provided on the **Answer Sheet: Gradescope MCQ**
- Part C contains 4 Short Answer Questions (SAQ) worth 36 marks.
- Answer all SAQs in the spaces provided on this paper using a pen. If you need additional writing space, please use the extra pages provided at the end of this exam booklet. Only pages in this exam booklet and the Gradescope MCQ sheet will be marked.

Please tick the box to confirm that your examination paper is complete. ☐

For Examiner Use Only

Q	Mark
1	
2	
3	
4	
5	
6	
7	
8	
9	
10	
11	
12	
13	
14	

Total_____

Instructions

- **Exam conditions:** This is a pen-and-paper based closed book exam.
- Final exam consists of 14 questions.

	Question type	Points	Recommended time spent
Question 1-6	MCQ	1 x 6	10 minutes
Question 7 - 10	Analyzing Code and figuring out the output	2 x 4	20 minutes
Question 11	Writing Code from specification	9 x 1	25 minutes
Question 12	Find errors in the given code	9 x 1	15 minutes
Question 13-14	Writing Code from specification	9 x 2	50 minutes

Practice Exam Available on Canvas

Review Material

Find the number of errors assuming it only takes positive integers as input.

Specify the line numbers and mention the corrections required.

```
public class PrintMax {
  public static void main(String args) {
        int a = args.size();
        int max = 0;
        for (int i; i < a; ++i) {
         if (args[i] > max)
            max = args[i];
         else
             max = max;
        }
        System.out.println(max);
    }
}
```

Review Material

Find the number of errors assuming it only takes positive integers as input.

Specify the line numbers and mention the corrections required

```
1. public class PrintMax {
2.   public static void main(String args) {        → public static void main(String[] args) {
3.       int a = args.size();                       → int a = args.length;
4.       int max = 0;
5.       for (int i; i < a; ++i) {                  → for (int i = 0; i < a; ++i) {
6.        if (args[i] > max)                        → if (Integer.parseInt(args[i]) > max)
7.           max = args[i];                         →      max = Integer.parseInt(args[i]);
8.        else
9.           max = max;
10.      }
11.      System.out.println(max);
12.   }
13. }
```

Review Material

Create an abstract class ***DiscountPolicy***. It should have an abstract method *computeDiscount* that will return the totalCost after discount for the purchase of a given number of a single item. The method has two int parameters, *numberOfItems* and *perItemPrice*.

Derive a class **BulkDiscount** from *DiscountPolicy*. It should have a constructor that has two parameters, *minimum* and *percent*. It should define the method computeDiscount so that if the quantity purchased of an item is more than minimum, the discount is <*percent*> percent.

Review Material

Create an abstract class DiscountPolicy. It should have a single abstract method computeDiscount that will return the totalCost after discount for the purchase of a given number of a single item. The method has two int parameters, *numberOfItems* and *perItemPrice*.

Derive a class BulkDiscount from DiscountPolicy. It should have a constructor with two parameters, *minimum* and *percent*. It should define the method computeDiscount so that if the quantity purchased of an item is more than minimum, the discount is <*percent*> percent.

```
abstract class DiscountPolicy{
   public abstract double computeDiscount(int numberOfItems, double perItemPrice);
}

public class BulkDiscount extends DiscountPolicy{
   int minimum;
   double percent;

   public BulkDiscount(int minimum, double percent){
      this.minimum = minimum;
      this.percent = percent;
   }

   public double computeDiscount(int numberOfItems, double perItemPrice){
      double totalCost = numberOfItems * perItemPrice;
      if(numberOfItems > minimum)
         totalCost -= (totalCost * percent)/100;

      return totalCost;
   }
}
```

Exception class

Create a **Shop** class which contains two methods:

- addProduct(int productID, int numberOfItems)
  - It adds a new product and stores it in a HashMap
- sell(int productID, int numberOfItemSold)
  - It reduces the numberOfItems with the numberOfItemSold.
    - If the productID is incorrect, print an appropriate error message.
    - If the numberOfItemSold is greater than numberOfItems left, throw an InsufficientItemException. You have to create this checked exception class called InsufficientItemException which calls its parent constructor with String “Insufficient number of items for <productID>”

Exception class

```
class InsufficientItemException extends Exception{
   public InsufficientItemException(int productID){
      super("Insufficient number of items for "+productID);
   }
}

public class Shop{

   HashMap<Integer, Integer> products;

   public Shop(){
      products = new HashMap<>();
   }

   void addProduct(int productID, int numberOfItems){
      products.put(productID, numberOfItems);
   }

   void sell(int productID, int numberOfItemSold) throws InsufficientItemException{
      if(!products.containsKey(productID))
                System.out.println("Incorrect product id: " +productID);
      else{
         int amount = products.get(productID);
         if(numberOfItemSold > amount)
                throw new InsufficientItemException(productID);
         else
                products.replace(productID, amount - numberOfItemSold);
      }
   }
}
```

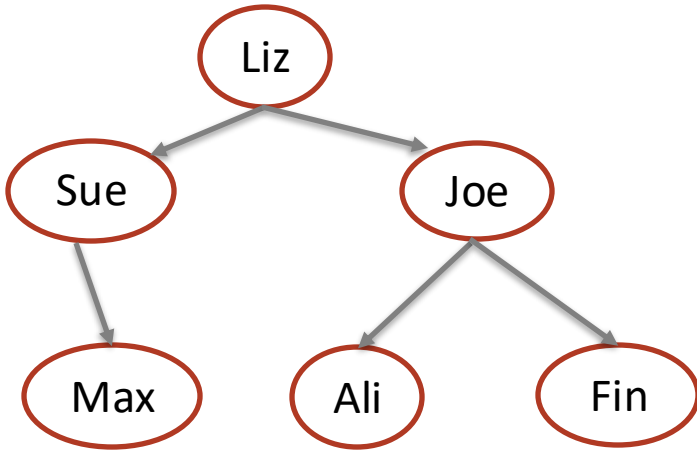
Create a **Shop** class which contains two methods:

- addProduct(int productID, int numberOfItems)
  - It adds a new product and stores it in a HashMap
- sell(int productID, int numberOfItemSold)
  - It reduces the numberOfItems with the numberOfItemSold.
    - If the productID is incorrect, print an appropriate error message.
    - If the numberOfItemSold is greater than numberOfItems left, throw an InsufficientItemException. You have to create this checked exception class called InsufficientItemException which calls its parent constructor with String "Insufficient number of items for <productID>"

Review Material

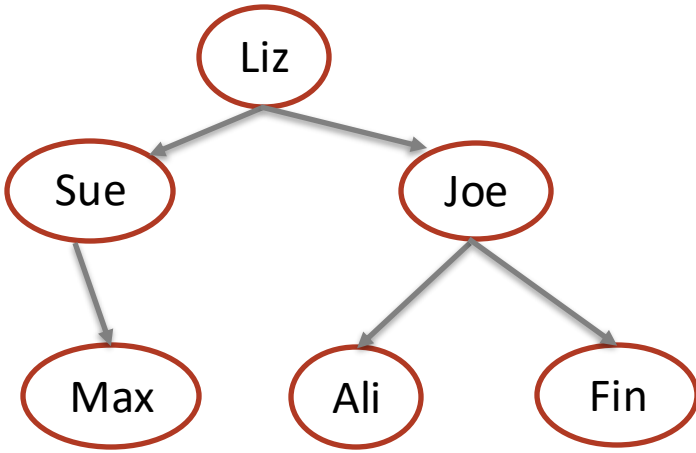
Consider the following FamilyMember class. Write a recursive method to count the number of leaf nodes (family members with no children) in the family tree.

```
class FamilyMember {

   String name;
   List<FamilyMember> children;

   public FamilyMember(String name) {
         this.name = name;
         children = new ArrayList<>();
   }

   public void addChildren(FamilyMember f){
         children.add(f);
   }

   public int countLeaf() {
         //your implementation here
   }
}
```



Review Material

```
class FamilyMember {
    String name;
    List<FamilyMember> children;

    public FamilyMember(String name) {
        this.name = name;
        children = new ArrayList<>();
    }
    public void addChildren(FamilyMember f){
        children.add(f);
    }
    public int countLeaf() {
        int counter = 0;
        if(this.children.size() == 0)
            return 1;
        else{
            for(FamilyMember f : children)
                counter += f.countLeaf();
        }
        return counter;
    }
}
```



Recursion

Suppose we have a satellite in orbit. To communicate to the satellite, we can send messages composed of two signals: dot and dash. Dot takes 2 microseconds to send, and dash takes 3 microseconds to send. Imagine that we want to know the number of different messages, M(k), that can be sent in k microseconds.

- If k is 0 or 1, we can't send a message
- If k is 2 or 3, we can send 1 message (dot or dot/dash, respectively)
- If k is larger than 3, we know that the message can start with either dot or dash.
    - If the message starts with dot, the number of possible messages is M(k – 2).
    - If the message starts with dash, the number of possible messages is M(k – 3).

So, the number of messages that can be sent in k microseconds is M(k – 2) + M(k – 3).

Write a program that reads a value of k from the keyboard and displays the value of M(k), which is computed by a recursive method.

Recursion

```
public class Satellite{

  long countMessage(int k){
          if(k < 2)
                return 0;
          else if(k == 2 || k == 3)
                return 1;
          else
               return countMessage(k-2) + countMessage(k-3);
   }
}
```

Improve the time efficiency of your program using the memoisation technique.

- If k is 0 or 1, we can't send a message
- If k is 2 or 3, we can send 1 message (dot or dash, respectively)
- If k is larger than 3, we know that the message can start with either dot or dash.
    - If the message starts with dot, the number of possible messages is M(k – 2).
    - If the message starts with dash, the number of possible messages is M(k – 3).

So, the number of messages that can be sent in k microseconds is M(k – 2) + M(k – 3).

Recursion

```
public class Satellite{

    long countMessage(int k){
      if(k < 2)                    return 0;
      else if(k == 2 || k == 3) return 1;
      else
          return countMessage(k-2) + countMessage(k-3);
    }
}
```

Recursion

```
public class Satellite{

   HashMap<Integer, Long> map = new HashMap<>();

   long countMessageMem(int k){

     if(map.containsKey(k)) return map.get(k);
     else if(k < 2)                return 0;
     else if(k == 2 || k == 3) return 1;
     else{
         long n = countMessageMem(k-2) + countMessageMem(k-3);
         map.put(k, n);
         return n;
      }
   }

   long countMessage(int k){
     if(k < 2)                     return 0;
     else if(k == 2 || k == 3) return 1;
     else
         return countMessage(k-2) + countMessage(k-3);
   }
}
```

Anonymous class / Lambda Expression

Anonymous Class	Lambda Expression
It is a class without name.	It is a method without name. (anonymous function)
It is the best choice if we want to handle interface with multiple methods.	It is the best choice if we want to handle functional interface.
At the time of compilation, a separate .class file will be generated.	At the time of compilation, no separate .class file will be generated. It simply convert it into private method of the outer class.
Memory allocation is on demand, whenever we are creating an object.	It resides in a permanent memory of JVM.

What is the expected output?

```java
interface GenericInterface<T> {
    T myMethod(T params);
}

public class MyClass {

    public static void main(String[] args) {

        GenericInterface<String> myString = new MyClass1();
        System.out.println(myString.myMethod("desserts"));

        GenericInterface<Integer> myInteger = new MyClass2();
        System.out.println(myInteger.myMethod(5));
    }
}

class MyClass1 implements GenericInterface<String> {

    public String myMethod(String str) {
        String result = "";
        for (int i = str.length() - 1; i >= 0; i--) {
            result += str.charAt(i);
        }
        return result;
    }
}
```

```java

class MyClass2 implements GenericInterface<Integer> {

    public Integer myMethod(Integer n) {
        int result = 1;
        for (int i = 1; i <= n; i++) {
            result = i * result;
        }
        return result;
    }
```

Anonymous/Lambda

```java
interface GenericInterface<T> {
    T myMethod(T params);
}

public class MyClass {

    public static void main(String[] args) {

        GenericInterface<String> myString = new MyClass1();
        System.out.println(myString.myMethod("desserts"));

        GenericInterface<Integer> myInteger = new MyClass2();
        System.out.println(myInteger.myMethod(5));
    }
}

class MyClass1 implements GenericInterface<String> {

    public String myMethod(String str) {
        String result = "";
        for (int i = str.length() - 1; i >= 0; i--) {
            result += str.charAt(i);
        }
        return result;
    }
}
```

```java

class MyClass2 implements GenericInterface<Integer> {

    public Integer myMethod(Integer n) {
        int result = 1;
        for (int i = 1; i <= n; i++) {
            result = i * result;
        }
        return result;
    }
```

Anonymous class

```java
interface GenericInterface<T> {
    T myMethod(T params);
}

public class MyAnonymousClass {

    public static void main(String[] args) {

        GenericInterface<String> myString = new GenericInterface<String>(){
            public String myMethod(String str){
                String result = "";
                for (int i = str.length() - 1; i >= 0; i--) {
                result += str.charAt(i);
                }
                return result;
            }
        };
        System.out.println(myString.myMethod("desserts"));

        GenericInterface<Integer> myInteger = new GenericInterface<Integer>() {
            public Integer myMethod(Integer n){
                int result = 1;
                for (int i = 1; i <= n; i++) {
                    result = i * result;
                }
                return result;
            }
        };
        System.out.println(myInteger.myMethod(5));
    }
}
```

Lambda Expression

```
1  interface GenericInterface<T> {
2      T myMethod(T params);
3  }
4
5  public class MyLambdaClass {
6
7      public static void main(String[] args) {
8
9          GenericInterface<String> myString = (str) -> {
10             String result = "";
11             for (int i = str.length() - 1; i >= 0; i--) {
12             result += str.charAt(i);
13             }
14             return result;
15         };
16         System.out.println(myString.myMethod("desserts"));
17
18         GenericInterface<Integer> myInteger = (n) ->{
19             int result = 1;
20             for (int i = 1; i <= n; i++) {
21                 result = i * result;
22             }
23             return result;
24         };
25         System.out.println(myInteger.myMethod(5));
26     }
27 }
```

Generics and Wildcards

Can we assign inherited types with generics? **NO.**

However, we are able to **read** super types using wildcards and **write** to a list knowing its lower bound.

Wildcards

Given some class that utilises generics, we are able to specify a wildcard by using **?** symbol. This will allow the many different types to be associated with the container.

Syntax:

Type<?> variable;

Type<? super LowerBound> variable;

Type<? extends UpperBound> variable;

Example:

```
List<?> list;
List<? extends Person> people;
List<? super Employee> employees;
```

Generics and Wildcards

```
FruitBasket<Fruit> b1 = new FruitBasket<Apple>();
```
a

```
FruitBasket<Apple> b2 = new FruitBasket<Fruit>();
```
a

```
FruitBasket<? super Apple> b3 = new FruitBasket<Apple>();
b3.setFruit(new Apple());
```
c

```
FruitBasket<? extends Orange> b4 = new FruitBasket<Orange>();
b4.setFruit(new Orange());
```
a

```
FruitBasket b5 = new FruitBasket();
b5.setFruit(new Orange());
```
b

For the above code snippets, identify whether the code:

a) fails to compile,

b) compiles with a warning or

c) compiles and runs without error

```
class Fruit{

}

class Apple extends Fruit {

}

class Orange extends Fruit {

}

public class FruitBasket<E> {
  private E fruit;
  public void setFruit(E x) {
     fruit = x;
  }

  public E getFruit() {
     return fruit; }
}
```

… and that's it.

In future

General TIPs

- Stay curious and keep learning
- Contribute to open-source projects!
  - Practice and explore
  - Build reusable code
- Morden languages (e.g., GoLang, Rust)!

Best of luck!

THE UNIVERSITY OF SYDNEY