INFO1113 / COMP9003
Object-Oriented Programming

Lecture 12

These slides will be available on ed 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**
- **Content Review**

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

Exam Topics

- Class inheritance
- Text File I/O
- Interfaces and abstract classes
- Collections
- Enums
- Recursion
- Generics, Type Bounds, and Wildcards
- Overloading and Overriding
- Exceptions
- JUnit Testing
- Anonymous class / Lambda Expression

Review Material (True/False)

1. When we define a class, we have also created a datatype — True
2. Primitive types can be assigned to null — False
3. Arrays are primitive types — False
4. ArrayLists are of fixed length — False
5. LinkedLists keep elements in arbitrary positions of memory — True

Review Material

Specify whether the followings are valid:

Q1 Invalid `Wolf w = new DomesticDog();`

Q2 Valid `Animal a1 = new DomesticDog();`

Q3 Valid `Animal a2 = new Wolf();`

Q4 Invalid `DomesticDog d = new Animal();`

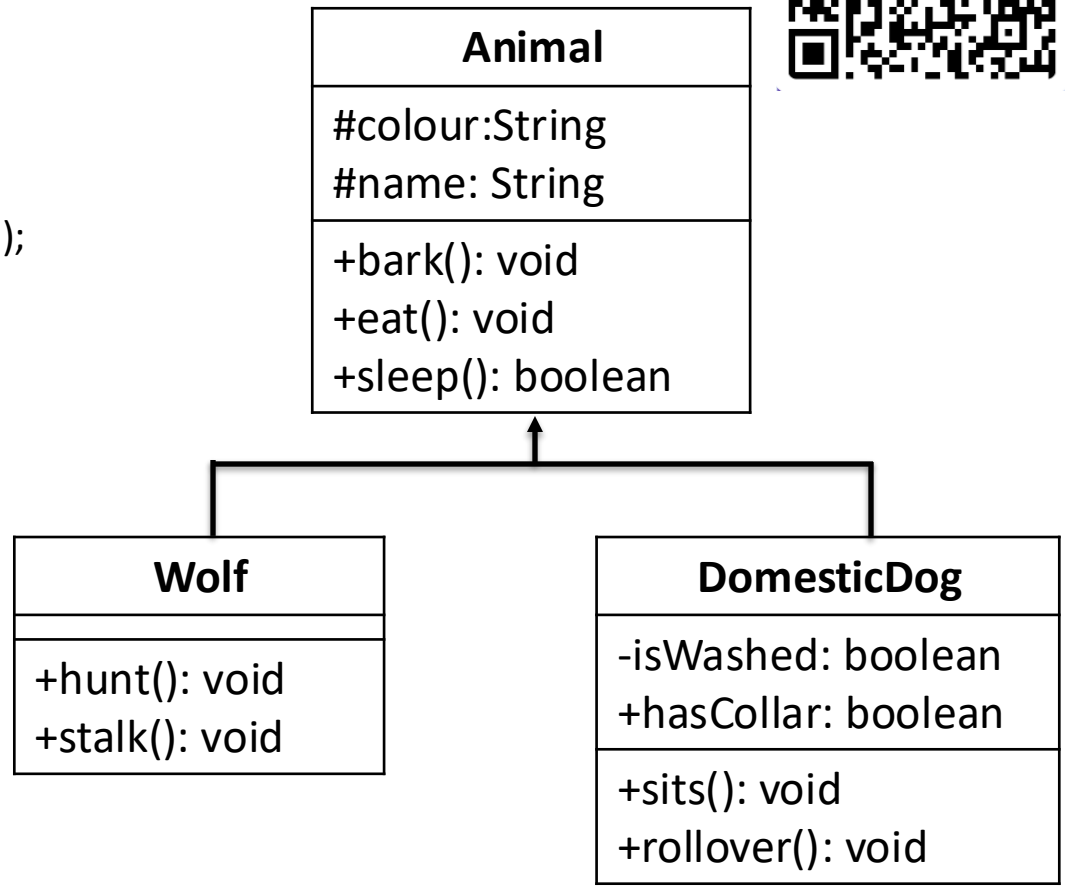
Q5 Invalid `a1.hunt();`

Q6 Invalid `a2.stalk();`

Q7 Invalid `a1.sits();`

Q8 Valid `a1.bark();`

Q9 Valid `a2.sleep();`



Review Material

Programming Question

Write a program to print the name of all the highest paid employees that has the following requirements:

The program will take in (as command-line arguments) pairs of inputs representing the name and salary of an employee.

The arguments will follow the pattern N_1 S_1 N_2 S_2 … for employees' name and salary as (N_1 , S_1), (N_2 , S_2) … and there could be any number of employees. You may assume that employee have one word name only.

These pairs of inputs should be stored in an array of Employee objects where Employee class has two attributes name and salary.

Review Material

```
class Employee{
   String name;
   double salary;
   public Employee(String name, double salary){
      this.name = name;
      this.salary = salary;
   }
}
```

Review Material

```
class Employee{
   String name;
   double salary;
   public Employee(String name, double salary){
      this.name = name;
      this.salary = salary;
   }
}
```

```
class HighestPaidEmployees{
   public static void main(String[] args) {
      Employee[] employees = new Employee[args.length/2];
      int numberOfEmployee = 0;
      String name = null;
      double maxSalary = 0;
      for(int i = 0; i < args.length; i++){
         if(i % 2 == 0)     name = args[i];
         else{
            double salary = Double.parseDouble(args[i]);
            employees[numberOfEmployee++] = new Employee(name, salary);
            if(salary > maxSalary)       maxSalary = salary;
         }
      }
      for(int i = 0; i < numberOfEmployee; i++)
         if(employees[i].salary == maxSalary)
            System.out.println(employees[i].name);
   }
}
```

Review Material

Given the following method declarations

1. `public void deduce(int x, int y)`
2. `public void deduce(int x, double y)`
3. `public void deduce(double x, double y)`
4. `public void deduce(String x, int y)`
5. `public void deduce(String x, Integer y)`

Specify which method will be invoked for each method call

`deduce(1, 2);` **method 1 (int x, int y)**

Review Material

Given the following method declarations

1. `public void deduce(int x, int y)`
2. `public void deduce(int x, double y)`
3. `public void deduce(double x, double y)`
4. `public void deduce(String x, int y)`
5. `public void deduce(String x, Integer y)`

Specify which method will be invoked for each method call

`deduce(1, 2);`	**method 1 (int x, int y)**
`deduce(2, 2.0);`	**method 2 (int x, double y)**

Review Material

Given the following method declarations

1. `public void deduce(int x, int y)`
2. `public void deduce(int x, double y)`
3. `public void deduce(double x, double y)`
4. `public void deduce(String x, int y)`
5. `public void deduce(String x, Integer y)`

Specify which method will be invoked for each method call

`deduce(1, 2);`	**method 1 (int x, int y)**
`deduce(2, 2.0);`	**method 2 (int x, double y)**
`deduce((double)3, 2);`	**method 3 (double x, double y)**

Review Material

Given the following method declarations

1. `public void deduce(int x, int y)`
2. `public void deduce(int x, double y)`
3. `public void deduce(double x, double y)`
4. `public void deduce(String x, int y)`
5. `public void deduce(String x, Integer y)`

Specify which method will be invoked for each method call

`deduce(1, 2);`	**method 1 (int x, int y)**
`deduce(2, 2.0);`	**method 2 (int x, double y)**
`deduce((double)3, 2);`	**method 3 (double x, double y)**
`deduce((int) 3.0, (int) 2.0);`	**method 1 (int x, int y)**

Review Material

Given the following method declarations

1. `public void deduce(int x, int y)`
2. `public void deduce(int x, double y)`
3. `public void deduce(double x, double y)`
4. `public void deduce(String x, int y)`
5. `public void deduce(String x, Integer y)`

Specify which method will be invoked for each method call

`deduce(1, 2);`	**method 1 (int x, int y)**
`deduce(2, 2.0);`	**method 2 (int x, double y)**
`deduce((double)3, 2);`	**method 3 (double x, double y)**
`deduce((int) 3.0, (int) 2.0);`	**method 1 (int x, int y)**
`deduce(Integer.parseInt("12"), 2);`	**method 1 (int x, int y)**

Review Material

Given the following method declarations

1. `public void deduce(int x, int y)`
2. `public void deduce(int x, double y)`
3. `public void deduce(double x, double y)`
4. `public void deduce(String x, int y)`
5. `public void deduce(String x, Integer y)`

Specify which method will be invoked for each method call

```
deduce(1, 2);                          method 1 (int x, int y)
deduce(2, 2.0);                        method 2 (int x, double y)
deduce((double)3, 2);                  method 3 (double x, double y)
deduce((int) 3.0, (int) 2.0);          method 1 (int x, int y)
deduce(Integer.parseInt("12"), 2);     method 1 (int x, int y)
deduce("42", 123);                     method 4 (String x, int y)
```

Review Material

Given the classes:

*Your task is to implement the **size** method which will traverse the list and print out the total number of elements in the list.*

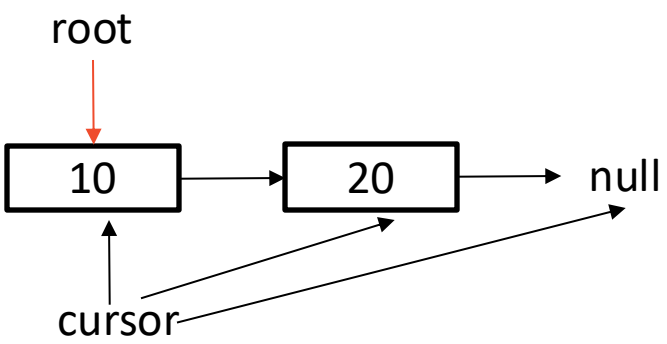
```
class Node<T>{
    public T element;
    public Node<T> next;

    public Node(T element){
        this.element = element;
        next = null;
    }
}
```

```
public class LinkedList<T>{
    Node<T> root;
    public LinkedList(){
        root = null;
    }
    public void add(T element){
        Node<T> newNode = new Node<T>(element);
        if(root == null)
            root = newNode;
        else{
            Node<T> cursor = root;
            while(cursor.next != null)
                cursor = cursor.next;
            cursor.next = newNode;
        }
    }

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

Review Material

Is this implementation correct?

```
class Node<T>{
    public T element;
    public Node<T> next;

    public Node(T element){
        this.element = element;
        next = null;
    }
}
```



count = 0 count = 1 return count

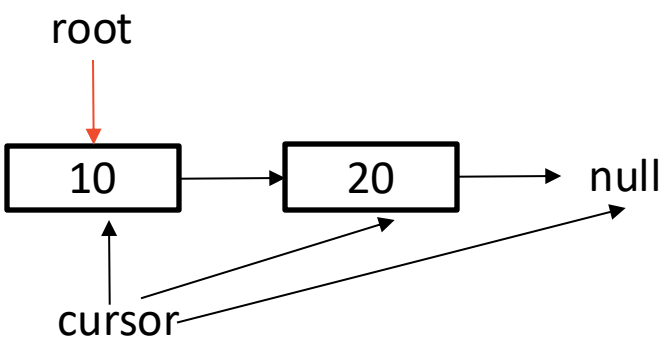
```
public class LinkedList<T>{
    Node<T> root;
    public LinkedList() { root = null;  }
    public void add(T element){
        Node<T> newNode = new Node<T>(element);
        if(root == null)    root = newNode;
        else{
            Node<T> cursor = root;
            while(cursor.next != null)
                cursor = cursor.next;
            cursor.next = newNode;
        }
    }
    public int size(){
        Node<T> cursor = root;
        int count = 0;
        while(cursor.next != null){
            ++count;
            cursor = cursor.next;
        }
        return count;
    }
}
```

Wrong implementation.
It will return the number of elements -1

Review Material

Is this implementation correct?

```
class Node<T>{
    public T element;
    public Node<T> next;

    public Node(T element){
        this.element = element;
        next = null;
    }
}
```



count = 1 count = 2 return count

```
public class LinkedList<T>{
    Node<T> root;
    public LinkedList() { root = null;  }
    public void add(T element){
        Node<T> newNode = new Node<T>(element);
        if(root == null)    root = newNode;
        else{
            Node<T> cursor = root;
            while(cursor.next != null)
                cursor = cursor.next;
            cursor.next = newNode;
        }
    }
    public int size(){
            Node<T> cursor = root;
            int count = 0;
            while(cursor.next != null){
                ++count;
                cursor = cursor.next;
            }
            return count;
        }
}
```

Correct implementation.

What will be the output of the following programs? Explain your answer.

Question 1

```
1  class Animal{
2
3      public Animal(){
4          System.out.println("Base Constructor");
5      }
6
7      public String toString(){
8          return "Inheritance test";
9      }
10 }
11
12
13 class Cat extends Animal{
14
15     public Cat(){
16         System.out.println("Derived Constructor");
17     }
18 }
19
20 public class Inheritance1 {
21
22     public static void main(String[] args) {
23
24         Cat c = new Cat();
25     }
26 }
```

Output:

Base Constructor
Derived Constructor

Question 2

```
1  class Base {
2
3      private void method1(){
4          System.out.println("method1");
5      }
6  }
7
8  class Derived extends Base {
9
10     public void method2(){
11         method1();
12         System.out.println("method2");
13     }
14 }
15
16 public class Inheritance2 {
17
18     public static void main(String[] args) {
19
20         Derived d = new Derived();
21         d.method2();
22     }
23 }
```

Output:

Compile error

Question 3

```
1  class Sweet{
2
3      protected void price(){
4          System.out.println("Sweet");
5      }
6  }
7
8  class Sugar extends Sweet{
9
10     private void price(){
11         super.price();
12         System.out.println("Sugar");
13     }
14 }
15
16
17 public class Inheritance3{
18
19     public static void main(String[] args){
20         Sweet su = new Sugar();
21         su.price();
22     }
23 }
```

Output:

Compile error

As per the rule of overriding, we cannot apply weaker access specifier over a stronger access specifier.
Note:
Public 1st stronger access specifier
Protected 2nd stronger access specifier
Default 3rd stronger access specifier
Private the most weaker access specifier

Enum

```
/** An enumeration of card suits. */
enum Suit
{
    CLUBS("black"), DIAMONDS("red"), HEARTS("red"),
    SPADES("black");

    private final String color;

    private Suit(String suitColor)
    {
        color = suitColor;
    }
    public String getColor()
    {
        return color;
    }
}
```

If **Suit cardSuit** = **Suit.SPADES**, what is returned by each of the following?

a. System.out.println(cardSuit.ordinal()) 3

b. System.out.println(cardSuit.equals(Suit.CLUBS)) false

c. System.out.println(cardSuit.getColor()) black

d. System.out.println(cardSuit) SPADES

See you next time!

THE UNIVERSITY OF SYDNEY