INFO1113 Object-Oriented Programming

Week 12A: Revision Part 1

Slides developed by Tyson Thomas

Copyright Warning

Topics

- Exam Format
- Topics to cover
- Example Revision Questions

Final Exam

- Date: 1st of December 2020
- Time: 9:00 AM Sydney time
- Duration: 130 Minutes
  - Reading time: 10 Minutes
  - Writing time: 120 Minutes
- New Canvas site
  - Final Exam for: INFO1113
  - Access no later than 7 days before the exam
- Everyone starts the exam at the same time
  - Only one attempt allowed
  - No late submission
- Exam adjustment is done by the exam office
  - Notification no later than 3 days before the exam

Question Type:

- MCQs (5 Questions → 10 marks)
  - Determine the correct output
  - True/False
  - Fill in the blanks
  - Single/Multiple choice
- Essay Type (5 Questions→40 marks)
  - Identify errors
  - Explain functionality
  - Write code

Examination Topics

- Simple class inheritance
- Interfaces and abstract classes
- UML Class Hierarchy Diagrams
- Instance and static variables
- Collections and Enums
- Recursion
- Wildcards
- Generics and Type Bounds
- Overloading and Overriding
- Testing

Review Material (True/False)

- A subclass can override from a method marked with final False

Review Material (True/False)

- A subclass can override from a method marked with final False
- When we define a class, we have also defined a type True

Review Material (True/False)

- A subclass can override from a method marked with final False
- When we define a class, we have also defined a type True
- Primitive types can be assigned to null False

Review Material (True/False)

- A subclass can override from a method marked with final False
- When we define a class, we have also defined a type True
- Primitive types can be assigned to null False
- Arrays are primitive types False

Review Material (True/False)

- A subclass can override from a method marked with final False
- When we define a class, we have also defined a type True
- Primitive types can be assigned to null False
- Arrays are primitive types False
- ArrayLists are fixed length False

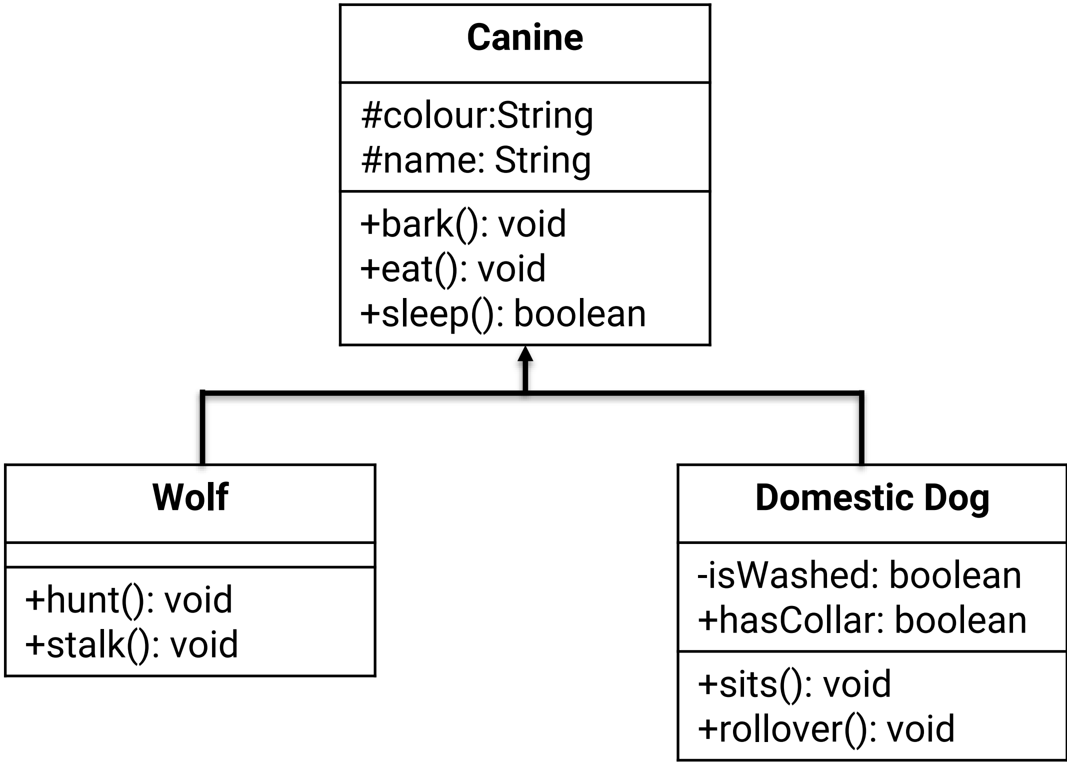
Review Material (True/False)

- A subclass can override from a method marked with final False
- When we define a class, we have also defined a type True
- Primitive types can be assigned to null False
- Arrays are primitive types False
- ArrayLists are fixed length False
- LinkedLists hold elements in arbitrary positions of memory True

Review Material

Specify which are valid

Invalid   Wolf w = new DomesticDog()

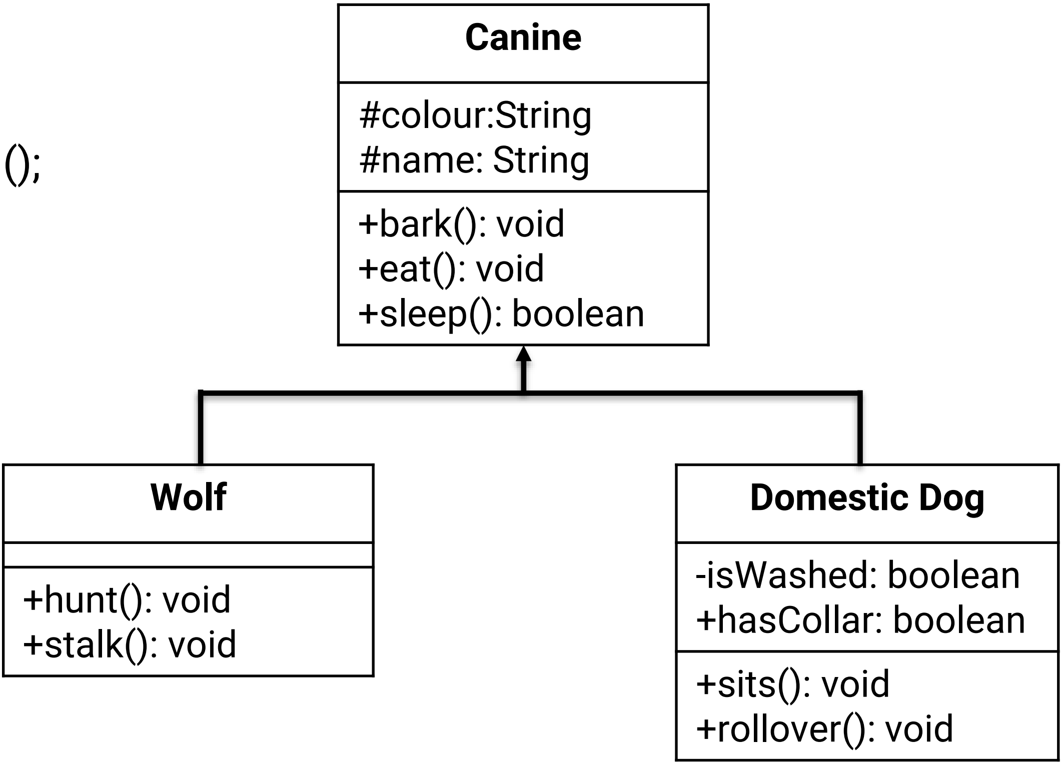


Review Material

Specify which are valid

Invalid Wolf w = new DomesticDog()

Valid Canine c = new DomesticDog();



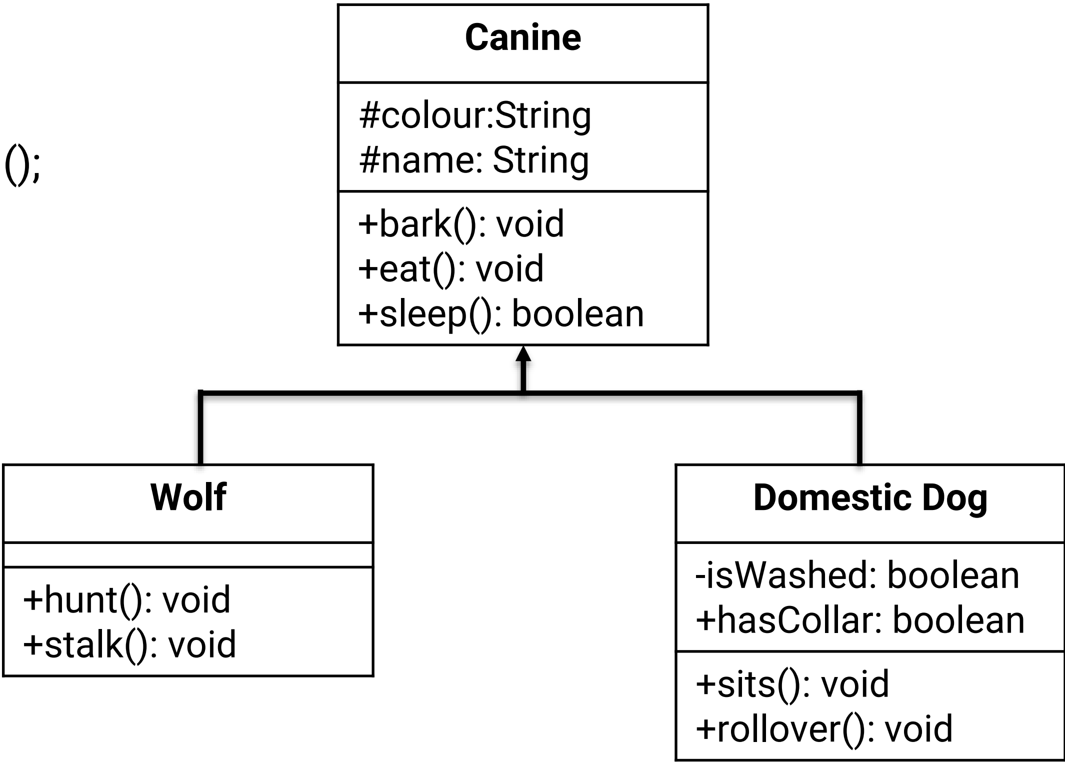
Review Material

Specify which are valid

Invalid Wolf w = new DomesticDog()

Valid Canine c = new DomesticDog();

Valid Canine d = new Wolf();



Canine

#colour:String
#name: String

+bark(): void
+eat(): void
+sleep(): boolean

Wolf

+hunt(): void
+stalk(): void

Domestic Dog

-isWashed: boolean
+hasCollar: boolean

+sits(): void
+rollover(): void

Review Material

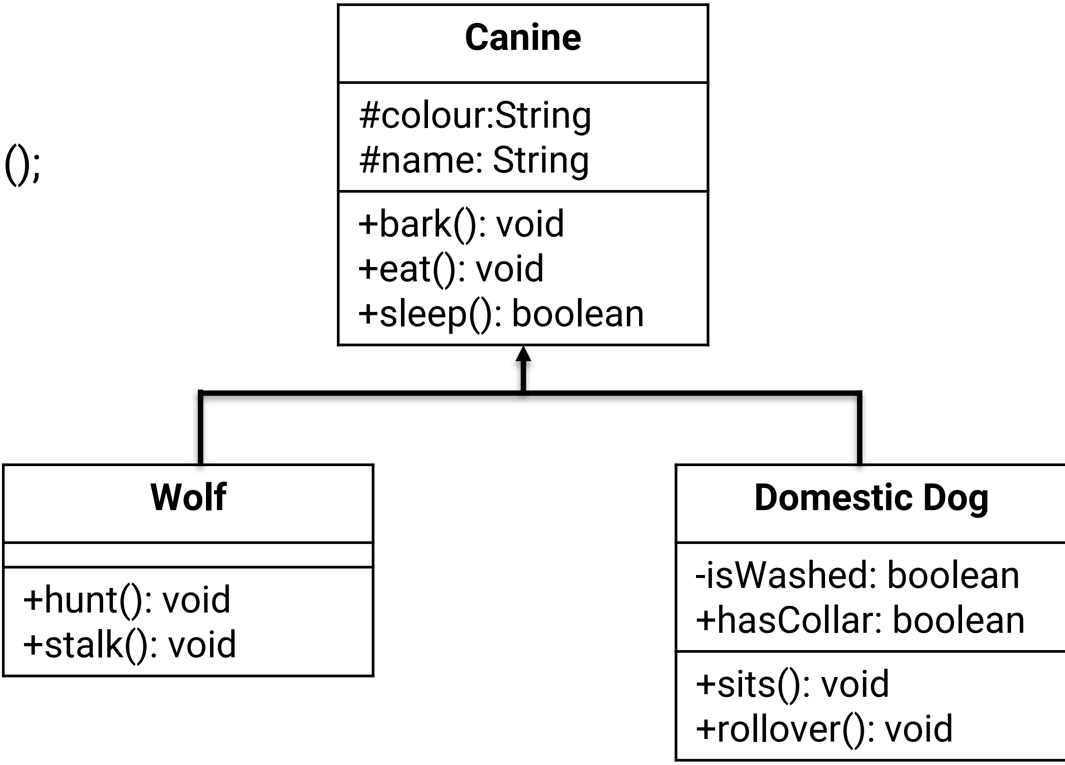
Specify which are valid

Invalid Wolf w = new DomesticDog()

Valid Canine c = new DomesticDog();

Valid Canine d = new Wolf();

Invalid c.hunt();



Canine

#colour:String
#name: String

+bark(): void
+eat(): void
+sleep(): boolean

Wolf

+hunt(): void
+stalk(): void

Domestic Dog

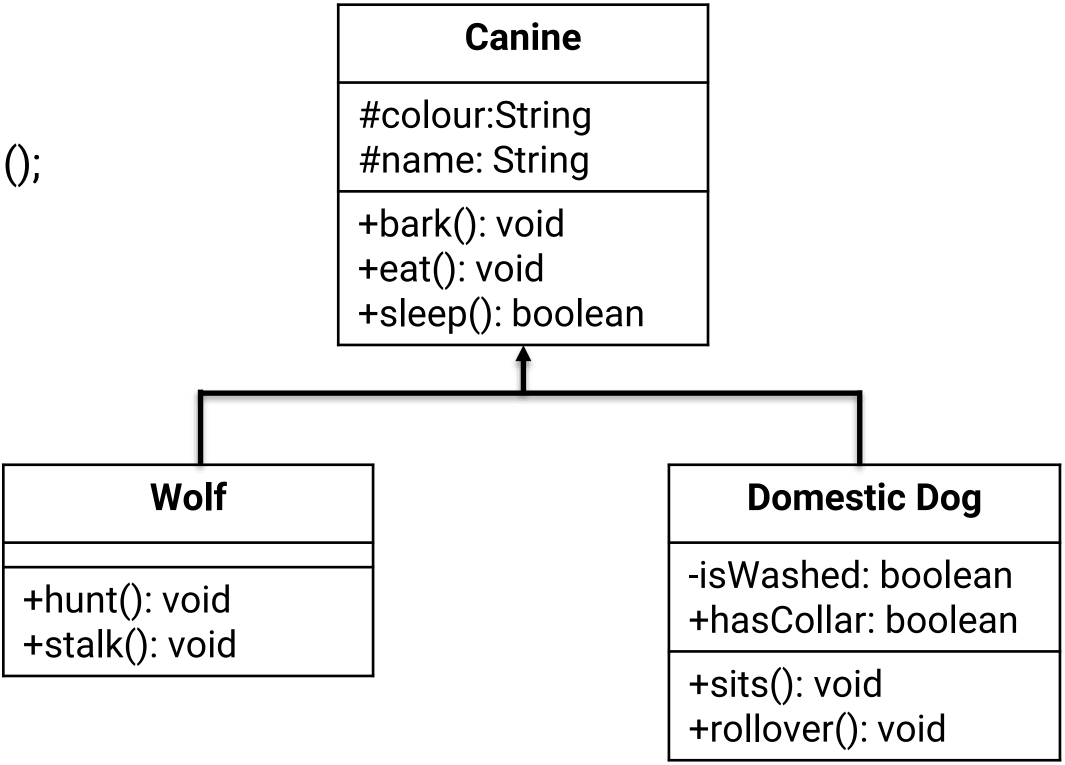
-isWashed: boolean
+hasCollar: boolean

+sits(): void
+rollover(): void

Review Material

Specify which are valid

Invalid	Wolf w = new DomesticDog()
Valid	Canine c = new DomesticDog();
Valid	Canine d = new Wolf();
Invalid	c.hunt();
Invalid	d.stalk();



Canine

#colour:String
#name: String

+bark(): void
+eat(): void
+sleep(): boolean

Wolf

+hunt(): void
+stalk(): void

Domestic Dog

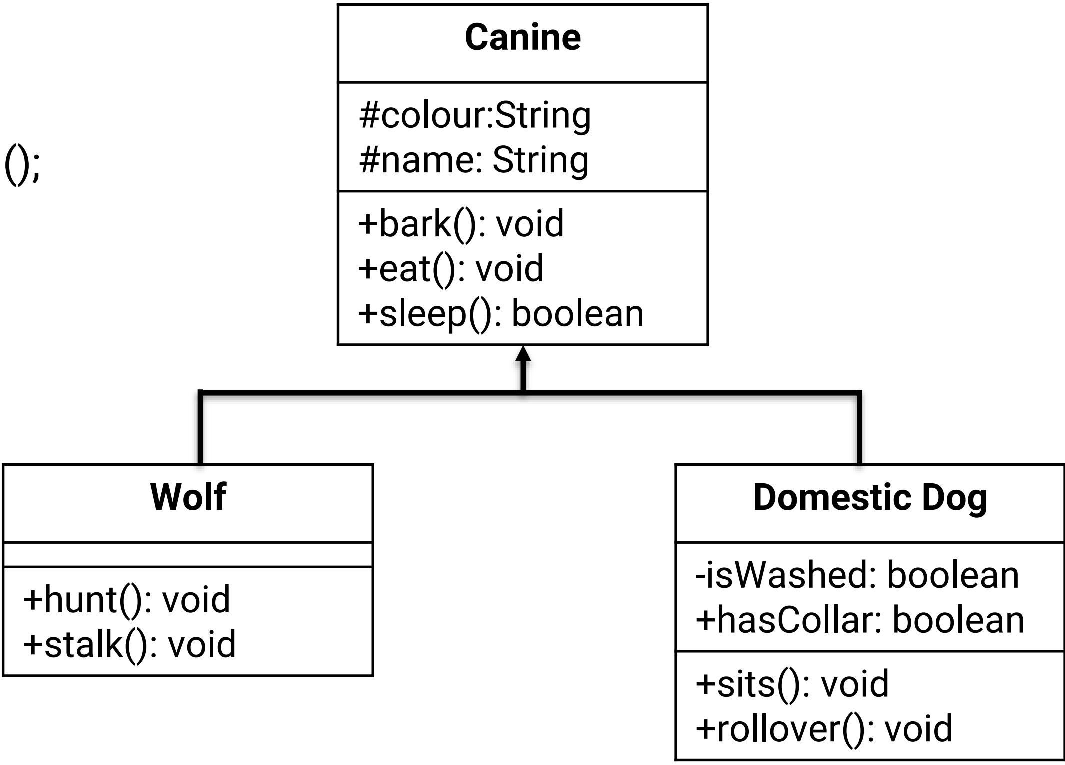
-isWashed: boolean
+hasCollar: boolean

+sits(): void
+rollover(): void

Review Material

Specify which are valid

Invalid	Wolf w = new DomesticDog()
Valid	Canine c = new DomesticDog();
Valid	Canine d = new Wolf();
Invalid	c.hunt();
Invalid	d.stalk();
Invalid	c.sits();



Canine

#colour:String
#name: String

+bark(): void
+eat(): void
+sleep(): boolean

Wolf

+hunt(): void
+stalk(): void

Domestic Dog

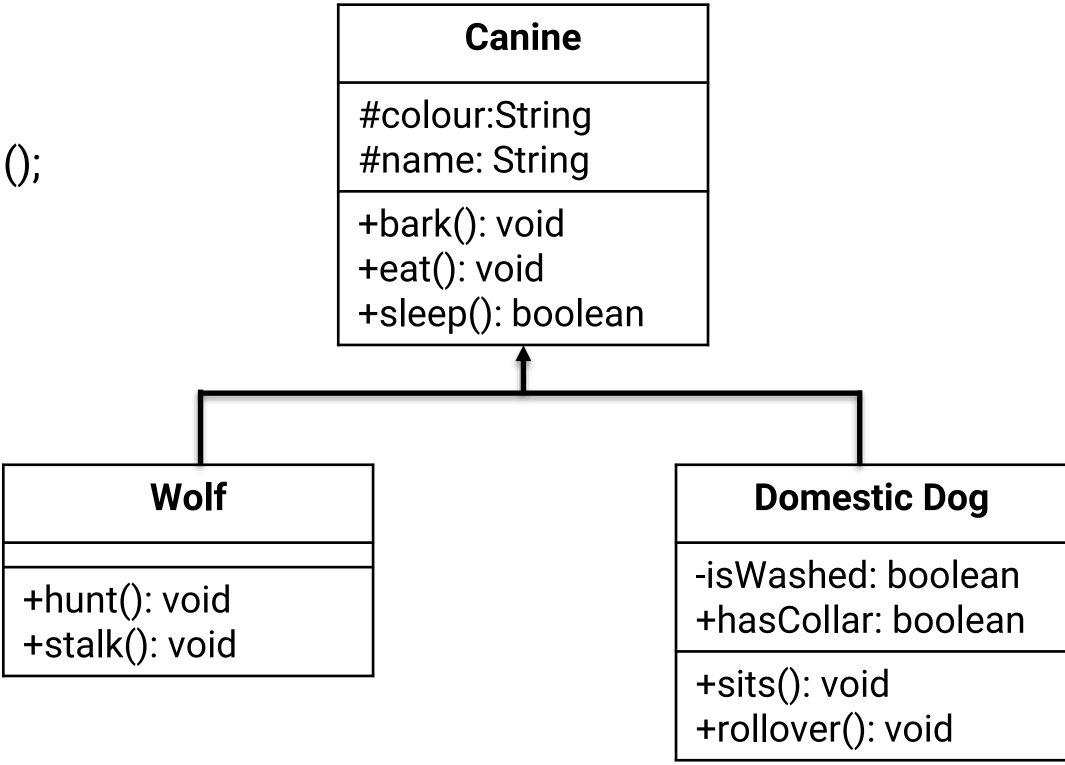
-isWashed: boolean
+hasCollar: boolean

+sits(): void
+rollover(): void

Review Material

Specify which are valid

Invalid	Wolf w = new DomesticDog()
Valid	Canine c = new DomesticDog();
Valid	Canine d = new Wolf();
Invalid	c.hunt();
Invalid	d.stalk();
Invalid	c.sits();
Valid	c.bark();



Canine

#colour:String
#name: String

+bark(): void
+eat(): void
+sleep(): boolean

Wolf

+hunt(): void
+stalk(): void

Domestic Dog

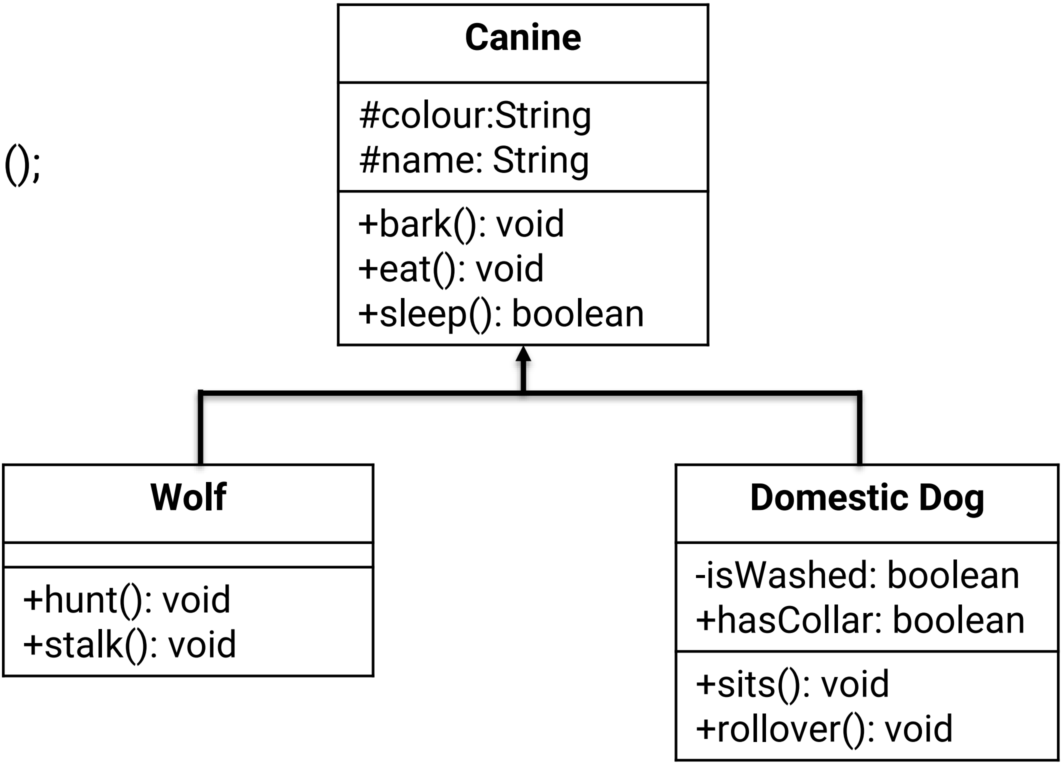
-isWashed: boolean
+hasCollar: boolean

+sits(): void
+rollover(): void

Review Material

Specify which are valid

Invalid	Wolf w = new DomesticDog()
Valid	Canine c = new DomesticDog();
Valid	Canine d = new Wolf();
Invalid	c.hunt();
Invalid	d.stalk();
Invalid	c.sits();
Valid	c.bark();
Valid	d.sleep();



Canine

#colour:String
#name: String

+bark(): void
+eat(): void
+sleep(): boolean

Wolf

+hunt(): void
+stalk(): void

Domestic Dog

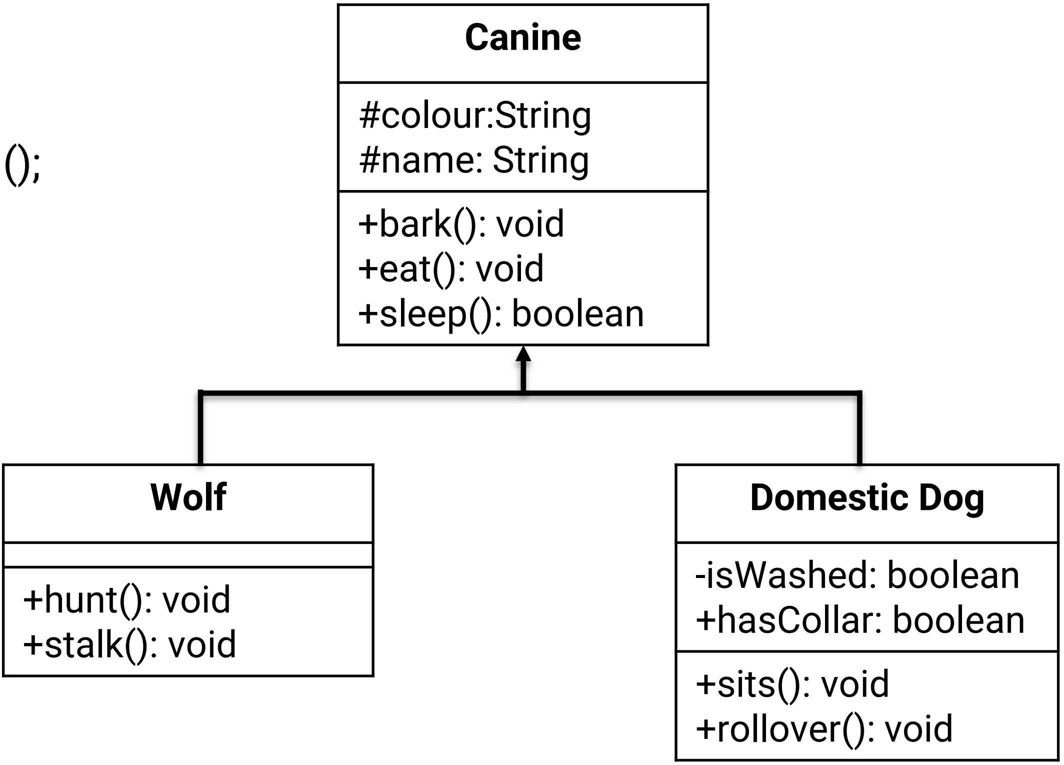
-isWashed: boolean
+hasCollar: boolean

+sits(): void
+rollover(): void

Review Material

Specify which are valid

Invalid	Wolf w = new DomesticDog()
Valid	Canine c = new DomesticDog();
Valid	Canine d = new Wolf();
Invalid	c.hunt();
Invalid	d.stalk();
Invalid	c.sits();
Valid	c.bark();
Valid	d.sleep();
Valid	c.eat();



Canine

#colour:String
#name: String

+bark(): void
+eat(): void
+sleep(): boolean

Wolf

+hunt(): void
+stalk(): void

Domestic Dog

-isWashed: boolean
+hasCollar: boolean

+sits(): void
+rollover(): void

Review Material

Programming Question

Write a program to find the highest paid employee 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

These pairs of inputs should be converted to an array of Employee objects.

Review Material

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

Review Material

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

```
class HighestPaidEmployee{
    public static void main(String[] args) {
        Employee[] employees = new Employee[args.length];
        int count = 0;
        String name = null;
        int maxSalary = 0;
        for(int i = 0; i < args.length; i++){
            if(i % 2 == 0)
                name = args[i];
            else{
                int salary = Integer.parseInt(args[i]);
                employees[count] = new Employee(name, salary);
                count++;
                if(salary > maxSalary)
                    maxSalary = salary;
            }
        }
        for(int i = 0; i < count; 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)

Specify what 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)

Specify what 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)

Specify what 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)

Specify what 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)

Specify what 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)

Specify what 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

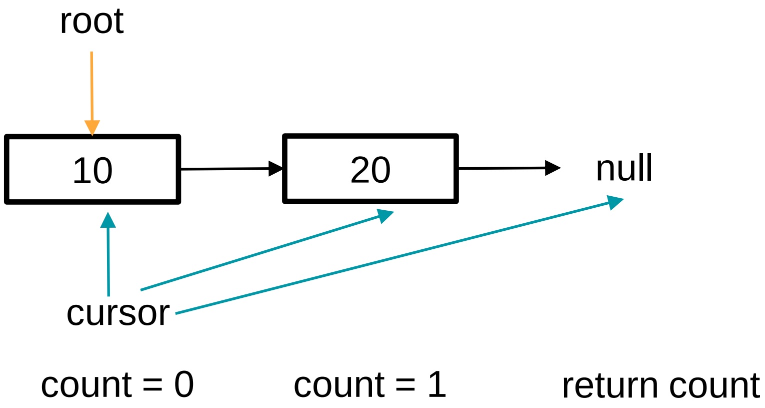
Is this implementation correct?

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

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

```
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;
   }
}
```

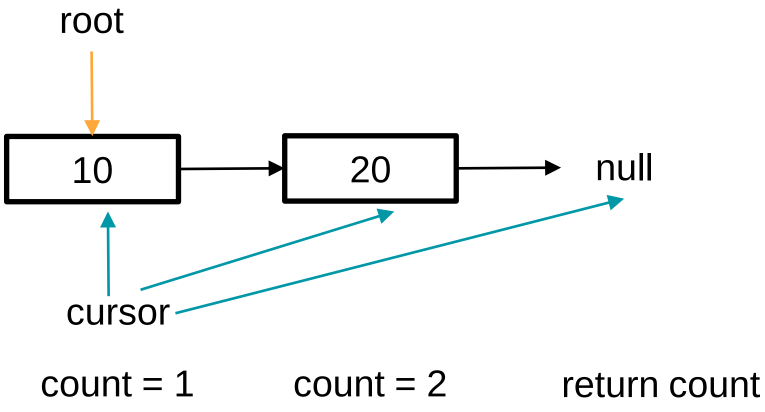
Wrong implementation.
It will return the
number of elements -1



Review Material

```
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(){
         Node<T> cursor = root;
         int count = 0;
         while(cursor.~~next~~ != null){
            ++count;
            cursor = cursor.next;
         }
         return count;
    }
}
```

Correct implementation

Suggestions

- You should have comprehensive notes, review them
- Review the lectures and tutorial materials
- Learn from all the mistakes you have made with your code
- If you haven’t attempted all **challenge** questions, attempt them to help with your review.
- Solve other problems using java and OOP

Unit of study survey

You have access to the unit of study survey

https://student-surveys.Sydney.edu.au/students/

Please respond to this survey as we are interested in what we can improve with this unit.

See you next time!