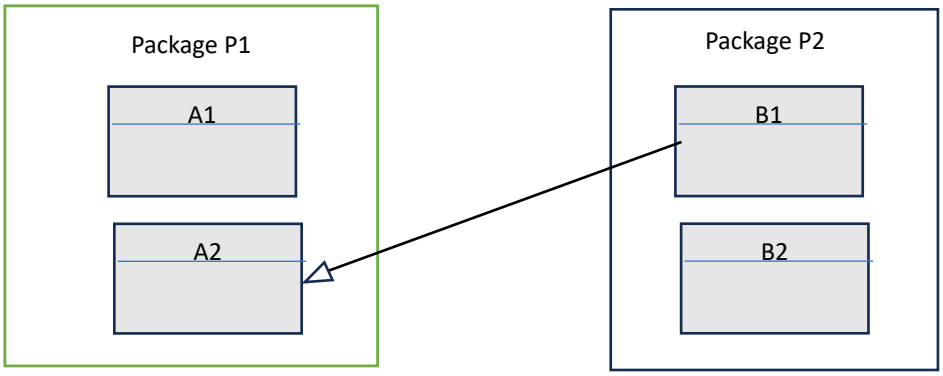
Final Exam Sample Questions

Short Answer questions (Access Modifiers)



Instance variables in A1 are private and methods are public. All members of A2 are protected. Answer the question based on the following scenario.

1) Which members of A1 can be accessed by A2?
2) Which members of A1 can be accessed by B1?
3) Which members of A2 can be accessed by B1?
4) Which members of A1 can be accessed by B2?
5) Which members of A2 can be accessed by B2?

Partial code (lambda expression):

Write a method randArray, which takes size as parameter. This method first fills an ArrayList of integers based on size with 0. Then use lambda expression to update the list using a random integer between [2, 10] and print the values on screen. Note: creating random number and print must happen in lambda expression.

Exception:

```
public class Exceptions {
    public static void main(String[] args) {
        System.out.print("A, ");
        mOne();
        System.out.print("B, ");
        try {
            mTwo(); // code below
            System.out.print("C, ");
            mThree();
            System.out.print("D, ");
        }
        catch (NullPointerException e)    {
            System.out.print("E, ");
        }
        catch (IllegalArgumentException e)    {
            System.out.print("F, ");
        }
```

```
        catch (Exception e)    {
            System.out.print("G, ");
            return;
        }
        finally {System.out.print("H, ");  }
        System.out.print("Z, ");
    }
    public static void mTwo() {
        System.out.println("I, ");
        mThree();
        System.out.println("J, ");
        try {
            System.out.println("K, ");
            mFour();
            System.out.println("L, ");
        }
        catch (NullPointerException e)    {
            System.out.println("M, ");
            throw new ArithmeticException();
        }
        finally { System.out.println("End of mTwo!"); }
    }//mTwo
}
```

Scenario	Description	Write the output and "Stack Trace" if one is printed
1	mOne: normal mThree: NullPointerException mdFour: NullPointerException	
2	mOne: IllegalArgumentException mThree: normal mdFour: NullPointerException	
3	mOne: normal mThree: normal mdFour: NullPointerException	
4	mOne: normal mThree: normal mdFour: IllegalArgumentException	

Refactoring – Given the following code for **Animal**, refactor it such that common functionality/data is lifted into abstract parent classes. Be sure to use data encapsulation principles. Assume that any private instance variable will need a getter method. Only add a setter method for a variable if needed. Show your answer as a UML diagram in the next page.

```
public enum AnimalKind { WHALE,BIRD,BAT}

    public class Animal {
      public AnimalKind kind;
```

```
    public Point position;
    public double weight;
    public double airSpeed;
    public double wingSpan;
    public double maxDepth;
    public Animal(AnimalKind kind, Point position, double weight,
          double airSpeed, double wingSpan, double maxDepth) {
            this.kind = kind;
            this.position = position;
            this.weight = weight;
            this.airSpeed = airSpeed;
            this.wingSpan = wingSpan;
            this.maxDepth = maxDepth;
    }
    public void getFood() {
            switch (kind) {
                    case WHALE:
                      getFoodWhale();
                      break;
                    case BLUE_BIRD:
                      getFoodBlueBird();
                      break;
                    case BAT:
                      getFoodBat();
          break;
            }
    }
    public void getFoodWhale() { /* code not shown */ }
    public void getFoodBlueBird() { /* code not shown */ }
    public void getFoodBat() { /* code not shown */ }
    public void fly(Point toLocation)  {
      if (kind == WHALE) {
        throw new UnsupportedOperationException();
      }
      if (kind == BAT) {  useSonarBat();       }
      position = toLocation;
    }
    public void useSonarBat() { /* code not shown */ }
    public boolean diveWhale(double depth) { position.z = depth;
    // other code not shown }
  }
```

Your UML Diagram will go here. Be sure to label any abstract classes or methods as <abstract> and interface as <interface>. As a refresher, the UML for the given code is shown below.

Animal
+ kind: AnimalKind + position: Point + weight: double + airSpeed: double + wingSpan: double + maxDepth: double
+ getFood() + getFoodWhale() + getFoodBlueBird() + getFoodBat() + fly() + useSonarBat() + diveWhale(): boolean

A* Algorithm: Below is a grid of an imaginary world. Squares in gray are obstacles. White squares are clear to travel through. The start and goal squares are labeled. Given the following partially constructed A Star table, complete all iterations of the algorithm. Make 26 the "current" square and add its neighbors to the table. Then choose the next valid "current" square and add its neighbors to the table. If there is more than one possible choice, choose whichever square using higher square number. Just show the first 5 moves.

- Be sure to add any squares to the closed list that need to be put there.
- Cross off any squares no longer on the open list.
- Use Manhattan distance for the heuristic.
- Use cardinal neighbors as your possible neighbors.

1	2	3	**4GOAL**	5	6
7	8	9	10	11	12
13	14	15	16	17	18
19	20	21	22	23	24
25	**26 START**	27	28	29	30
31	32	33	34	35	36

Open	Closed	g	h	f	Prior
26		0	6	6	-

Stream Questions:

Given the following Pet class, complete the two functions below. Your solution must use Streams. You may not write a loop in your code.

```
public class Pet {
	private String name;
	private int age;
	private String breed;
	private double weight;
	private char sex;

	public Pet(String name, int age, String breed, double weight, char sex)	{
		this.name = name;
		this.age = age;
		this.breed = breed;
		this.weight = weight;
		this.sex = sex;
	}

	public String name() { return name; }
	public int age() { return age; }
	public String breed() { return breed; }
	public double weight() { return weight; }
	public char sex() { return sex; }
}
```

1. In the Driver, complete the following functionality. Given a list of pets, use **streams** to return a new list with all the female "Labrador's" mapped to new pets with their age increased by one.
   public List<Pet> oldGals(List<Pet> pets){ …..
   }

2. In the Driver, complete the following functionality. Given a list of pets, use **streams** to compute and return the average weight of all the pets in the list.

   public double avgWeight(List<Pet> pets){ …..

   }

1. Create a Comparator for Pets that orders Pets in descending order by age. If there is a tie, Pets should be further ordered alphabetically by breed. ***For full credit, your solution must use a key extractor.***

2. Create a **Comparator** (no method just comparator) for Pets that orders Pets one years old or less first. In those subcategories, Pets should be further ordered in ascending order by weight. (Note: It is okay if your solution creates more than one Comparator. Name the Comparator that accomplishes the complete task: tinyComp). ***For full credit, your solution must use lambda expression.***