31.

Your task here is to implement a Java code based on the following specifications. Note that your code should match the specifications in a precise manner. Consider default visibility of classes, data fields, and methods unless mentioned otherwise.

Specifications:

```
class definitions:
  class Dog:
    data members:
     String name
     int age
     int weight

   Dog(String name, int age, int weight): constructor with public
visibility
    Define getter setters with public visibility
    toString(): has been implemented for you

  class Implementation:
     method definition:
       filterByAgeAndWeight(List<Dog> listDog):
           return type: List<Dog>
           visibility: public

       separateWithDelimeter(List<Dog> listDog):
           return type: String
           visibility: public
```

Task:

class Dog:

- Define the class according to the above specifications

class Implementation:

Implement the below method for this class:

- List<Dog> filterByAgeAndWeight(List<Dog> listDog): fetch dog details on the basis of:
- age greater than 10
- weight greater than 25
- get the filtered data, put it into a list and return the list

- String separateWithDelimeter(List<Dog> listDog): concat and return the dogs details with delimeter "$~$~"

Refer sample output for clarity

Sample Input

```
List<Dog> list = new ArrayList<Dog>();
 list.add(new Dog("German Shepherd ", 20, 35));
 list.add(new Dog("Labrador ", 5, 40));
 list.add(new Dog("Pitbull ", 29, 100));
 list.add(new Dog("Poodle", 10, 45));
```

Sample Output

```
[Dog{name='German Shepherd ', age=20, weight=35},
Dog{name='Pitbull ', age=29, weight=100}]

-------------------------------------------------------

Dog{name='German Shepherd ', age=20,
weight=35}$~$~Dog{name='Labrador ', age=5,
weight=40}$~$~Dog{name='Pitbull ', age=29,
weight=100}$~$~Dog{name='Poodle', age=10, weight=45}
```

NOTE

- You can make suitable function calls and use the RUN CODE button to check your main() method output.

ALLOWED TECHNOLOGIES

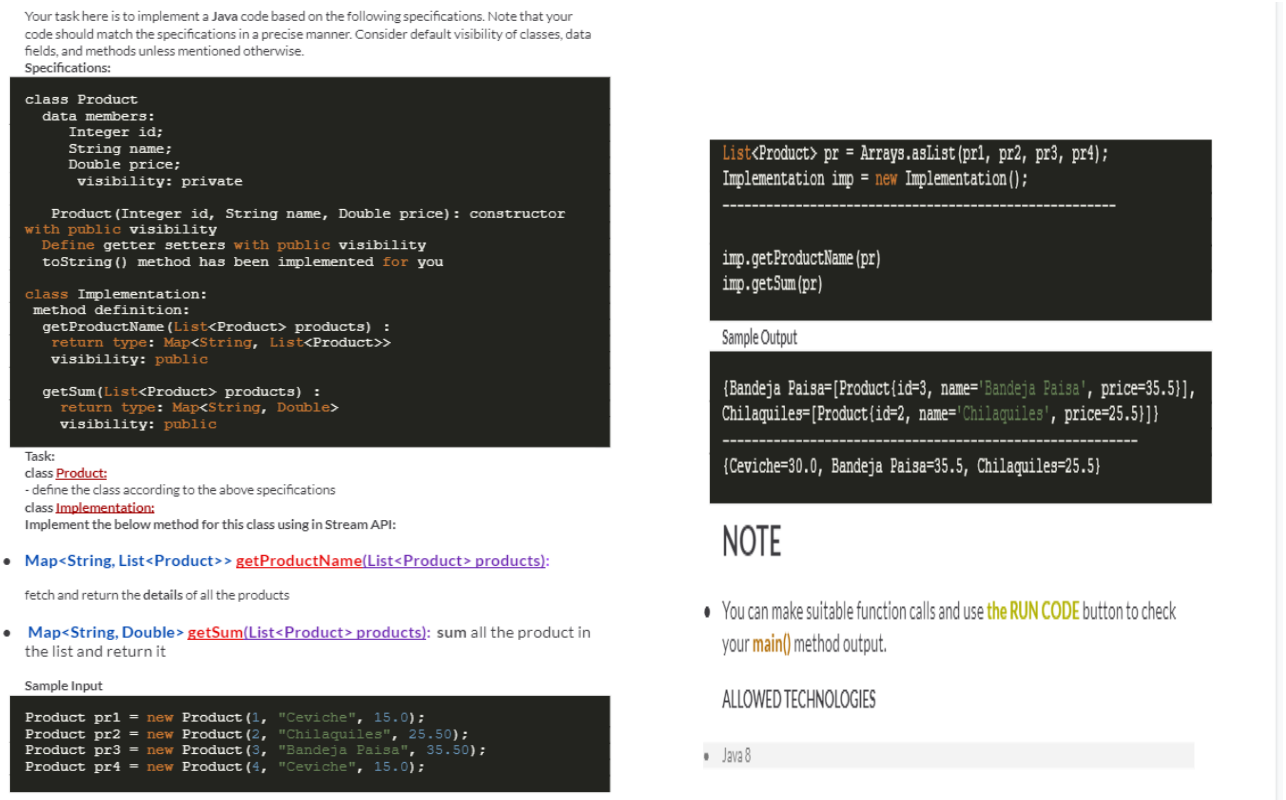
- Java 8

```
package q31;
import java.util.ArrayList;
import java.util.List;

public class Dog {
String name;
int age;
int weight;
public Dog(String name, int age, int weight) {
        super();
        this.name = name;
        this.age = age;
        this.weight = weight;
}
public String getName() {
        return name;
}
public void setName(String name) {
        this.name = name;
}
public int getAge() {
        return age;
}
public void setAge(int age) {
```

```
		this.age = age;
	}
	public int getWeight() {
		return weight;
	}
	public void setWeight(int weight) {
		this.weight = weight;
	}
	@Override
	public String toString() {
		return "Dog{name='" + name +"', age="+age+ ", weight=" + weight + "}";
	}
	public static void main(String[] args) {
		List<Dog> list = new ArrayList<Dog>();
		list.add(new Dog("German Shepherd ", 20, 35));
		list.add(new Dog("Labrador ", 5, 40));
		list.add(new Dog("Pitbull ", 29, 100));
		list.add(new Dog("Poodle", 10, 45));
		Implementation implementation=new Implementation();
		System.out.println(implementation.filterByAgeAndWeight(list));
		System.out.println(implementation.separateWithDelimeter(list));
	}
}
class Implementation{
   public List<Dog> filterByAgeAndWeight(List<Dog> listDog){
     int i=0;
     List<Dog> list=new ArrayList<>();
     for(Dog d:listDog) {
		if(listDog.get(i).getAge()>10 && listDog.get(i).getWeight()>25) {
			list.add(d);
		}
		i++;
     }
     return list;
   }
   public String separateWithDelimeter(List<Dog> listDog){
     StringBuffer sb=new StringBuffer();
		int n=listDog.size();
		sb.append(listDog.get(0));
		for(int i=1;i<n;i++) {
			sb.append("$$");
			sb.append(listDog.get(i));
		}
		return sb.toString();
   }
}
```

32.



Your task here is to implement a Java code based on the following specifications. Note that your code should match the specifications in a precise manner. Consider default visibility of classes, data fields, and methods unless mentioned otherwise.

Specifications:

```
class Product
  data members:
     Integer id;
     String name;
     Double price;
      visibility: private

   Product(Integer id, String name, Double price): constructor
with public visibility
  Define getter setters with public visibility
  toString() method has been implemented for you

class Implementation:
 method definition:
  getProductName(List<Product> products) :
   return type: Map<String, List<Product>>
   visibility: public

  getSum(List<Product> products) :
    return type: Map<String, Double>
    visibility: public
```

Task:

class Product:

- define the class according to the above specifications

class Implementation:

Implement the below method for this class using in Stream API:

- Map<String, List<Product>> getProductName(List<Product> products):

  fetch and return the details of all the products

- Map<String, Double> getSum(List<Product> products): sum all the product in the list and return it

Sample Input

```
Product pr1 = new Product(1, "Ceviche", 15.0);
Product pr2 = new Product(2, "Chilaquiles", 25.50);
Product pr3 = new Product(3, "Bandeja Paisa", 35.50);
Product pr4 = new Product(4, "Ceviche", 15.0);
```

```
List<Product> pr = Arrays.asList(pr1, pr2, pr3, pr4);
Implementation imp = new Implementation();
-------------------------------------------------


imp.getProductName(pr)
imp.getSum(pr)
```

Sample Output

```
{Bandeja Paisa=[Product{id=3, name='Bandeja Paisa', price=35.5}],
Chilaquiles=[Product{id=2, name='Chilaquiles', price=25.5}]}
-------------------------------------------------------
{Ceviche=30.0, Bandeja Paisa=35.5, Chilaquiles=25.5}
```

NOTE

- You can make suitable function calls and use the RUN CODE button to check your main() method output.

ALLOWED TECHNOLOGIES

- Java 8

Solution :

```
package q32;

import java.util.ArrayList;
import java.util.Arrays;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

public class Product {
	private Integer id;
	private String name;
	private Double price;
	public Product(Integer id, String name, Double price) {
		this.id = id;
		this.name = name;
		this.price = price;
	}
	public Integer getId() {
		return id;
	}
	public void setId(Integer id) {
		this.id = id;
	}
```

```
	public String getName() {
		return name;
	}
	public void setName(String name) {
		this.name = name;
	}
	public Double getPrice() {
		return price;
	}
	public void setPrice(Double price) {
		this.price = price;
	}
	@Override
	public String toString() {
		return "Product{id="+id+" name='" + name +"', price="+price+"}";
	}
	public static void main(String[] args) {
		Product pr1 = new Product(1, "Ceviche", 15.0);
		Product pr2 = new Product(2, "Chilaquiles", 25.50);
		Product pr3 = new Product(3, "Bandeja Paisa", 35.50);
		Product pr4 = new Product(4, "Ceviche", 15.0);
		List<Product> pr = Arrays.asList(pr1, pr2, pr3, pr4);
		Implementation implementation=new Implementation();
		System.out.println(implementation.getProductName(pr));
		System.out.println(implementation.getSum(pr));
	}
}
class Implementation{
	public Map<String, List<Product>> getProductName(List<Product> products){
		Map<String, List<Product>> tempMap = new HashMap<String, List<Product>>();
		for(Product p:products) {
			List<Product> ProductList=new ArrayList<>();
			if(p.getPrice()>20.0) {
				if(!tempMap.containsKey(p.getName())) {
					ProductList.add(p);
					tempMap.put(p.getName(),ProductList);
				}
				else {
					List<Product> temp=new ArrayList<>();
					ProductList.add(p);
					temp=tempMap.get(p.getName());
					temp.addAll(ProductList);
					tempMap.put(p.getName(),temp) ;
				}
			}
		}
		return tempMap;
	}
```

```
    public Map<String, Double> getSum(List<Product> products){
        Map<String, Double> tempMap = new HashMap<String, Double>();
        for(Product p:products) {
            if(!tempMap.containsKey(p.getName())) {
                tempMap.put(p.getName(),p.getPrice());
            }
            else {
                double temp=tempMap.get(p.getName());
                temp+=p.getPrice();
                tempMap.put(p.getName(),temp);
            }
        }
        return tempMap;
    }
}
```

33.

Your task here is to implement a Java code based on the following specifications. Note that your code should match the specifications in a precise manner. Consider default visibility of classes, data fields, and methods unless mentioned otherwise.

Specifications:

```
class Person
  data members:
     Integer id
     String name
     LocalDate birthDate
      visibility: private

   Person(Integer id, String name, LocalDate birthDate):
constructor with public visibility
  Define getter setters with public visibility
  toString() method has been implemented for you

class Implementation:
 method definition:
  filterListByBirth(List<Person> persons) :
   return type: List<Person>
   visibility: public

  limitSkipAndReturn(List<Person> persons, int pageNumber, int
pageSize) :
    return type: List<Person>
    visibility: public
```

Task:

class Person:

- define the class according to the above specifications

class Implementation:

Implement the below method for this class using in Stream API:

- List<Person> filterListByBirth(List<Person> persons): filter and return the list by date of birth
- List<Person> limitSkipAndReturn(List<Person> persons, int pageNumber, int pageSize): get the multiplication of pageNumber and pageSize, skip those indexes, now limit the pageSize and return the list

Example: For the below list in the sample input, page number * page size = 2, skip the first 2 indexes in the list, now limit the page size = 2, after limiting the page size we get the desired result as given below in sample output

Refer to the sample input output for more clarifications

Sample Input

```
Person p1 = new Person(1, "Mito", LocalDate.of(1991, 1, 21));
Person p2 = new Person(2, "Code", LocalDate.of(1990, 2, 21));
Person p3 = new Person(3, "Jaime", LocalDate.of(1980, 6, 23));
Person p4 = new Person(4, "Duke", LocalDate.of(2019, 5, 15));
Person p5 = new Person(5, "James", LocalDate.of(2010, 1, 4));
```

```
List<Person> persons = Arrays.asList(p1, p2, p3, p4, p5);
-------------------------------------------------------

imp.filterListByBirth(persons)
imp.limitSkipAndReturn(persons, 1, 2)
```

Sample Output

```
[Person{id=3, name='Jaime', birthDate=1980-06-23}, Person{id=2,
name='Code', birthDate=1990-02-21}, Person{id=1, name='Mito',
birthDate=1991-01-21}, Person{id=5, name='James',
birthDate=2010-01-04}, Person{id=4, name='Duke',
birthDate=2019-05-15}]
-------------------------------------------------------
[Person{id=3, name='Jaime', birthDate=1980-06-23}, Person{id=4,
name='Duke', birthDate=2019-05-15}]
```

NOTE

- You can make suitable function calls and use the RUN CODE button to check your main() method output.

ALLOWED TECHNOLOGIES

- Java 8

Solution :

```
package q33;

import java.time.LocalDate;
import java.util.ArrayList;
```

```
import java.util.Arrays;
import java.util.Collections;
import java.util.Comparator;
import java.util.List;

public class Person {
	private Integer id;
	private String name;
	private LocalDate birthDate;
	public Person(Integer id, String name, LocalDate birthDate) {
		this.id = id;
		this.name = name;
		this.birthDate = birthDate;
	}
	public Integer getId() {
		return id;
	}
	public void setId(Integer id) {
		this.id = id;
	}
	public String getName() {
		return name;
	}
	public void setName(String name) {
		this.name = name;
	}
	public LocalDate getBirthDate() {
		return birthDate;
	}
	public void setBirthDate(LocalDate birthDate) {
		this.birthDate = birthDate;
	}
	@Override
	public String toString() {
		return "Product{id="+id+" name='" + name +"', birthDate="+birthDate+"}";
	}
	public static void main(String[] args) {
		Person p1 = new Person(1, "Mito", LocalDate.of(1991, 1, 21));
		Person p2 = new Person(2, "Code", LocalDate.of(1990, 2, 21));
		Person p3 = new Person(3, "Jaime", LocalDate.of(1980, 6, 23));
		Person p4 = new Person(4, "Duke", LocalDate.of(2019, 5, 15));
		Person p5 = new Person(5, "James", LocalDate.of(2010, 1, 4));
		List<Person> persons = Arrays.asList(p1, p2, p3, p4, p5);
		Implementation imp=new Implementation();
		System.out.println(imp.filterListByBirth(persons));
		System.out.println(imp.limitSkipAndReturn(persons, 1, 2));
	}
}
```

```
class Implementation{
	public List<Person> filterListByBirth(List<Person> persons){
		List<Person> persons1=new ArrayList<>(persons);
		Collections.sort(persons1, Comparator.comparing(Person::getBirthDate));
		return persons1;
	}
	public List<Person> limitSkipAndReturn(List<Person> persons,int pageNumber,int
pageSize){
		List<Person> list=new ArrayList<>();
		int temp1=pageNumber*pageSize;
		int temp2=0;
		for(int i=temp1;i<persons.size();i++) {
			if(temp2==temp1)
				return list;
			list.add(persons.get(i));
			temp2++;
		}
		return list;
	}
}
```

34.

Complete the classes using the Specifications given below. Consider default visibility of classes, data fields, and methods unless mentioned otherwise.

Specifications

```
class definitions:
 class Bomb:
    data members:
        int time
        String color
    Bomb(int time, String color): constructor with public
visibility

class Suicide:
 data members:
     Bomb bomb

     Suicide(Bomb bb): constructor with public visibility

 method definitions:
     diffuseIt(int time, String color) throws Exception:
        return type: String
        visibility: public

     checkSafety(int time, String color) throws Exception:
        return type: String
        visibility: public

 class SuicideException extends Exception:
   method definition:
      SuicideException(String msg)
         visibility: public
```

Task

Class Bomb
- define the **int** variable **time**.
-define **String** variable **color**
-define a **constructor** according to the above specifications.

Class Suicide
-define the bomb variable with null;
-define a **constructor** according to the above specifications and initialize the bomb variable with the object passed in the argument.

Implement the below methods for this class:

-String diffuseIt(int time, String color) throws Exception:

- Write a code to validate the criteria for getting the award.
- **throw a SuicideException** if **time** is greater than **the time of the bomb** with the message "**Time exceeded**".
- **throw a SuicideException** if the **color of the bomb is different from the color passed as an argument** with the message "**Wrong color**".
- If no above exception is found then return a string message "**Hope is there**".

-String checkSafety(int time, String color) throws Exception:

- Write a code to send an invite to the nominee.
- If diffuseIt() method throws a **SuicideException** then returns a message "**Bomb exploded**".(Use try-catch block)
- If it throws any other exception then return a message "**Other exception**".
- If no exception is found then return a message "**Take a bow**".

Sample Input

```
Bomb b=new Bomb(10,"red");
Suicide sc=new Suicide(b);
String s = sc.diffuseIt(5,"red");
String t = sc.checkSafety(8,"blue");
s.toLowerCase();
t.toLowerCase();
```

Sample Output

```
hope is there
bomb exploded
```

NOTE:

- You can make suitable function calls and use **the RUN CODE** button to check your **main()** method output.

ALLOWED TECHNOLOGIES

- Java 8

Solution :

```
package doselect34;

public class Bomb {

	int time;
	String color;

	Bomb() {

	}

	Bomb(int time, String color) {
		this.time = time;
		this.color = color;
	}

	public void setTime(int time) {
		this.time = time;
	}

	public void setColor(String color) {
		this.color = color;
	}

	public int getTime() {
		return this.time;
	}

	public String getColor() {
		return this.color;
	}



class Suicide {

		Bomb bomb=null;
		public Suicide(Bomb bb) {
			bomb=bb;
		}
		public String diffuseIt(int time , String color) throws Exception{
			int bombtime = bomb.getTime();
			String bombcolor = bomb.getColor();

			if(time  > bombtime)
				throw new SuicideException("Time exceeded");
```

```
                if(color != bombcolor)
                        throw new SuicideException("Wrong color");
                else
                        return "Hope is there";

        }

        public String checkSafety(int time , String color) throws Exception{

                try {
                        diffuseIt(time , color);

                }
                catch(SuicideException e) {
                        return "Bomb exploded";
                }
                catch(Exception e) {
                        return "Other Exception";
                }
                return "take a bow";
        }
    }

    @SuppressWarnings("serial")
public class SuicideException extends Exception {

    public SuicideException(String msg) {
            super(msg);
            }
}
    public static void main(String[] args) throws Exception {
            // TODO Auto-generated method stub

            Bomb b = new Bomb(10, "red");
            Suicide sc = b.new Suicide(b);

             try {
                     System.out.println(sc.diffuseIt(5,"red").toLowerCase());

             }
             catch(SuicideException e){
                     System.out.println(e.getMessage().toLowerCase());
             }
             String t = sc.checkSafety(8,"blue");
                     System.out.println(t.toLowerCase());
    }

}
```

35.

Complete the classes using the Specifications given below. Consider default visibility of classes, data fields, and methods unless mentioned otherwise.

Specifications

```
class definitions:
 class Task:
  data members:
    String name
    int hours
  Task(String name, int hours): constructor with public
visibility

class ToDoList:
data members:
   List<Task> tasks

 method definitions:
   addTask(Task t) throws Exception:
    return type: String
    visibility: public

   completeTask(Task t) throws Exception:
    return type: String
    visibility: public

 class TaskException extends Exception:
  method definition:
   TaskException(String msg)
     visibility: public
```

Task

Class Task
- define the String variable name.
- define the int variable hours
-define a constructor according to the above specifications.

Class ToDoList

Implement the below methods for this class:

-String addTask(Task t) throws Exception:

- Write a code to validate the criteria for getting the award.
- throw a **TaskException** if '**hours**' is less than 1 or greater than 24 with the message "**Invalid time**".
- throw a **TaskException** if the **given object** is already present in the ArrayList with the message "**Already present**".
- If no above **exception** is found then add the given task to the given ArrayList and return a string message "**Task will be completed**".

-String completeTask(Task t) throws Exception:

Write a code to complete the task.

- If addTask() method throws a TaskException then returns a message "**Task incomplete**".(Use try-catch block)
- If it throws any other exception then return a message "**Other exception**".
- If no exception is found then return a message "**Task completed**".

Sample Input

```
Task t= new Task("Gym",12);
ToDoList lst=new ToDoList();
String t1=lst.addTask(t);
t1.toLowerCase();
```

Sample Output

```
task will be completed
```

NOTE:

- You can make suitable function calls and use the RUN CODE button to check your main() method output.

ALLOWED TECHNOLOGIES

- Java 8

Solution :

```
package doselect_q35;

import java.util.ArrayList;
import java.util.List;

public class Task {
	String name;
	int hours;

	Task(String name, int hours){
		this.name=name;
		this.hours=hours;
	}
	public String getName() {
		return  this.name;
	}
	public int getHours() {
		return this.hours;
	}
	public void setNme(String name) {
		this.name=name;
```

```
	}
	public void setHours(int hours) {
		this.hours=hours;
	}

	public static void main(String[] args) throws Exception{
		Task t = new Task("Gym",12);
		ToDoList list=new ToDoList();

		try {
			System.out.println(list.addTask(t).toLowerCase());
		}
		catch(Exception e) {
			System.out.println(e.getMessage().toLowerCase());
		}

	}

}


class ToDoList{
	List<Task> tasks = new ArrayList<>();


	public String addTask(Task t) throws Exception{
		int hour = t.getHours();
		if(hour <1 || hour >24) {
			throw new TaskException("Invalid time");
		}
		if(tasks.contains(t)) {
			throw new TaskException("Already Present");
		}
		else {
			tasks.add(t);
			return "Task will be completed";
		}
	}

	public String completeTask(Task t) throws Exception{
		try {
			addTask(t);
		}
		catch(TaskException e) {
			return "Task incomplete";
		}
		catch(Exception e) {
			return "Other Exception";
```

```
                }
                return "Task completed";
        }
}
```

ii)

```
package doselect_q35;

@SuppressWarnings("serial")
public class TaskException extends Exception {
        public TaskException(String msg) {
                super(msg);
        }

}
```