

Câu 3.

Như chúng ta đã biết, sức khỏe tốt là nền tảng góp phần cho một cuộc sống hạnh phúc, và việc tập luyện hàng ngày chính là một biện pháp cực kỳ hiệu quả để nâng cao cả về sức khỏe tinh thần và thể chất. Năm bắt xu thế quan tâm sức khỏe của người dân, ngày 26/10/2018 vừa qua, trung tâm thể hình MTK của thị xã Dĩ An đã được khai trương và bắt đầu đón nhận khách hàng.

Mỗi khách hàng tham gia CLB của trung tâm có thể tùy chọn một trong 3 gói dịch vụ khác nhau là Premium (nâng cao), Basic (cơ bản), Non-member (không thành viên). Đây là phương thức mà trung tâm tiến hành để từng loại đối tượng khách hàng có thể dễ dàng tiếp cận các dịch vụ tương ứng.

Bên cạnh thể hình là dịch vụ mặc định, tùy gói dịch vụ mà khách hàng có thể đăng kí tham gia các lớp học được cung cấp xuyên suốt các thời gian trong ngày như: Yoga, Aerobic, Boxing, múa bụng, body combat..., dịch vụ xông hơi và hỗ trợ từ huấn luyện viên cá nhân (PT) cũng sẽ được cung cấp.

Mỗi khách hàng khi đăng kí sẽ phải điền thông tin họ tên và số CMND và lựa chọn gói dịch vụ cũng như thời gian muốn sử dụng dịch vụ đó (theo tháng).

Chi tiết từng gói dịch vụ trong **một tháng** được mô tả theo bảng sau. Đơn vị: Nghìn đồng.

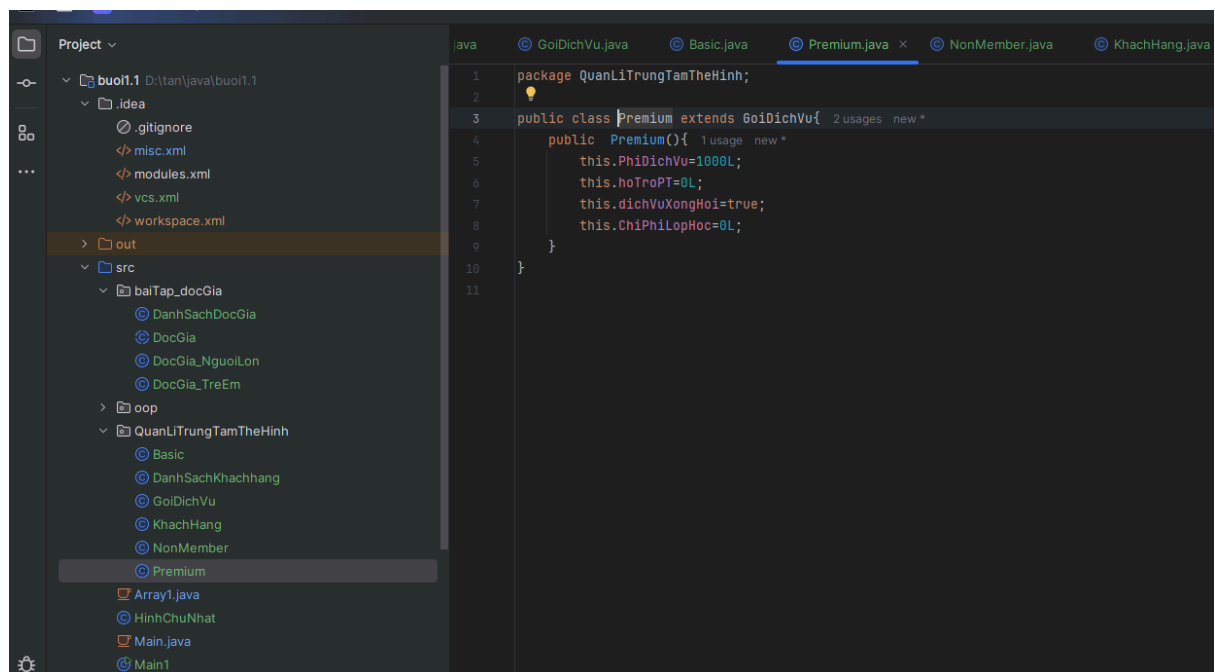
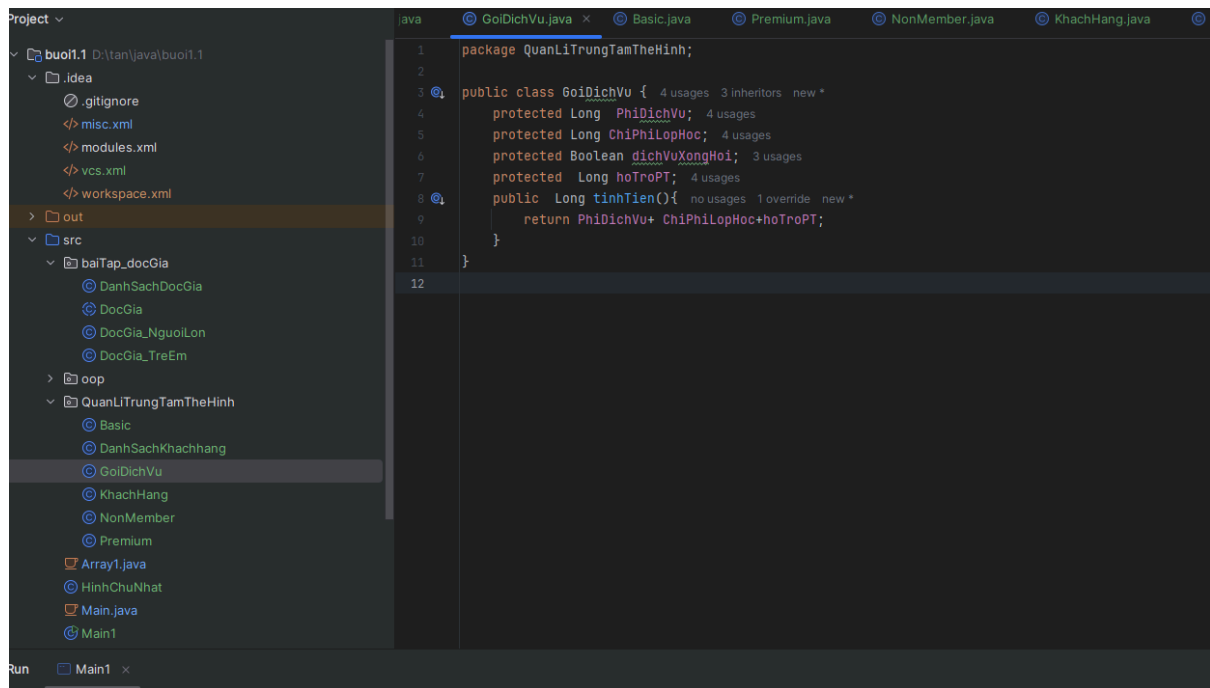
	Premium	Basic	Non-member
Phí cơ bản	1 000	500	200
Chi phí lớp học	Miễn phí	100 / lớp	Không có
Dịch vụ xông hơi	Miễn phí	Không có	Không có
Hỗ trợ PT	Miễn phí	100	200

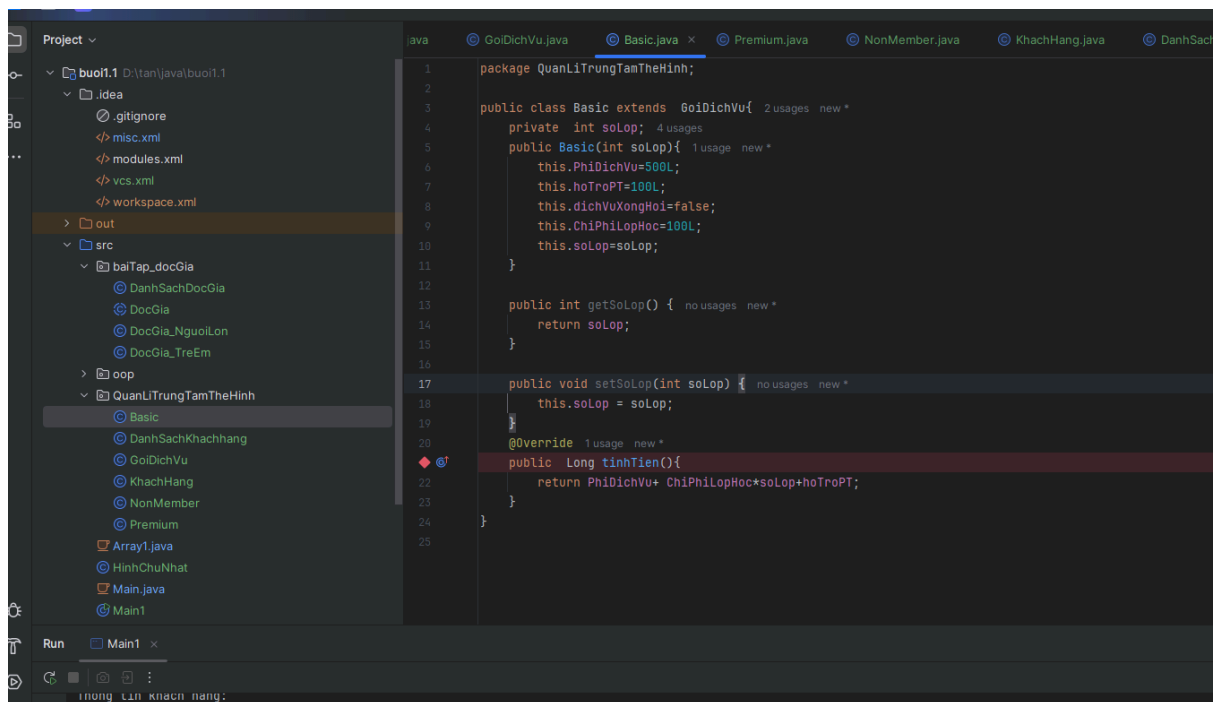
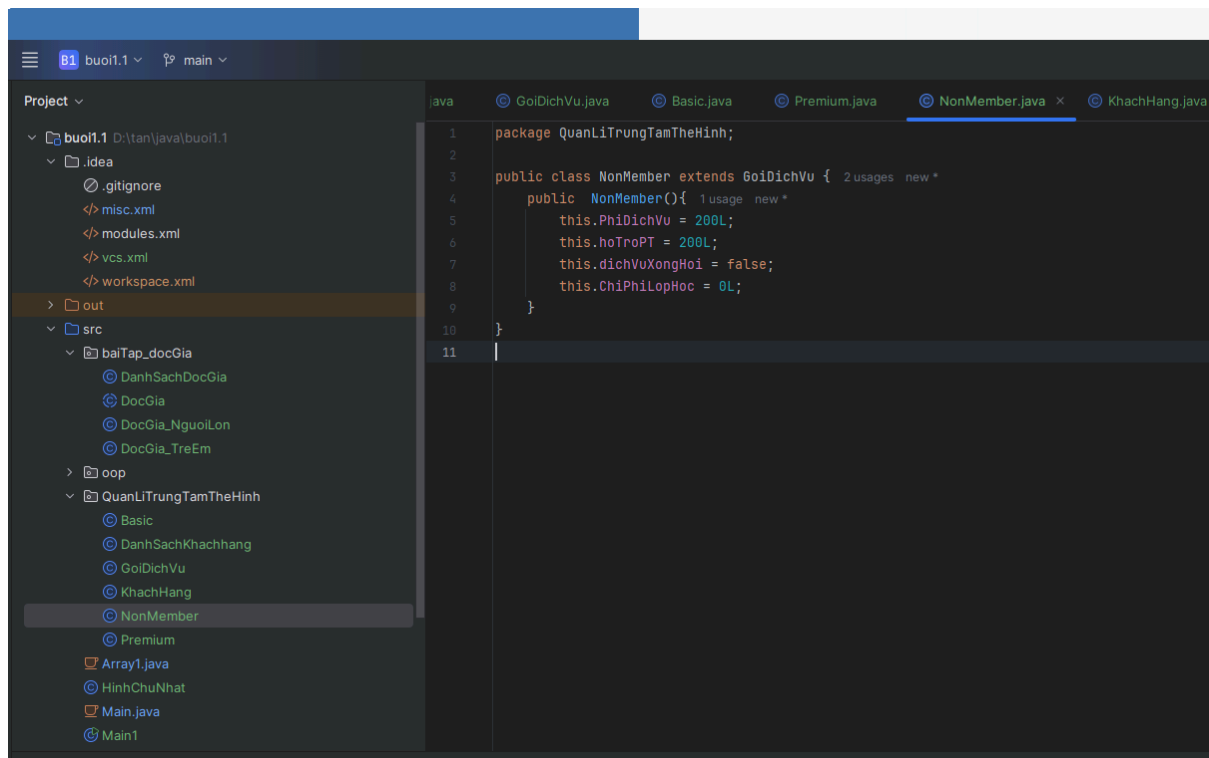
Lưu ý: *Miễn phí: Khách hàng không chỉ trả thêm tiền khi đăng kí tính năng này.*

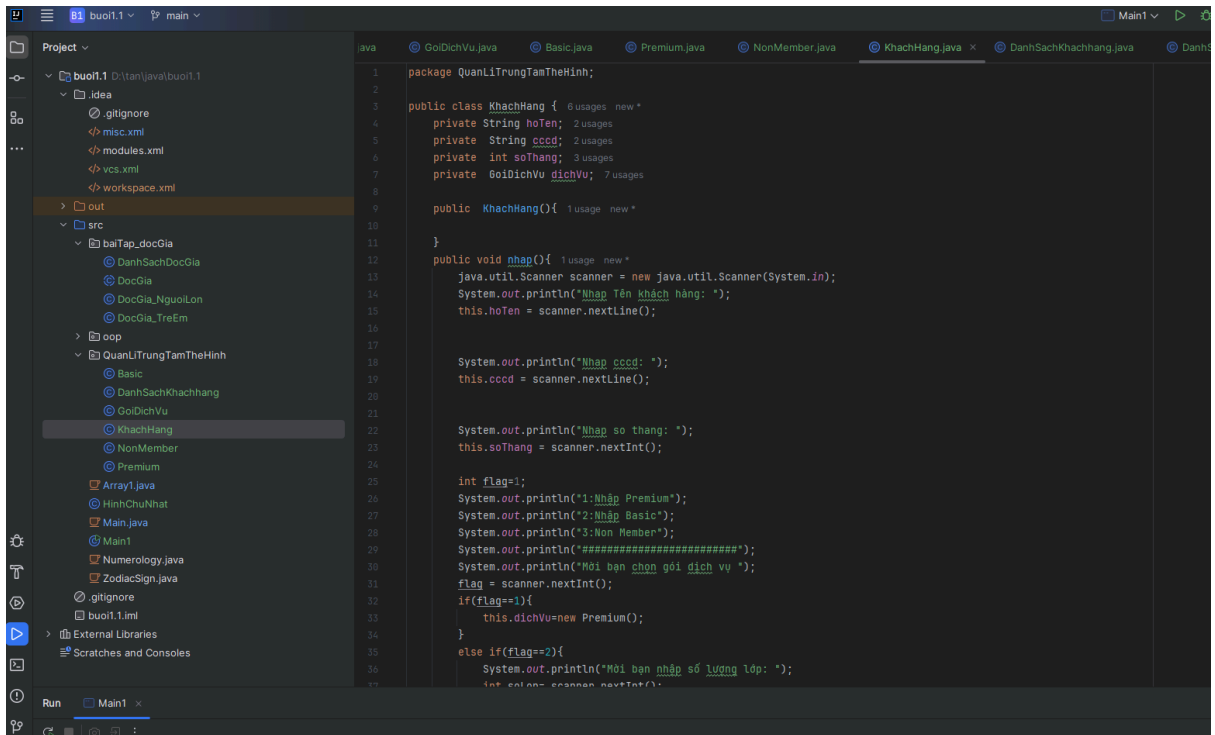
Không có: Khách hàng sẽ không được cung cấp tính năng này.

Sinh viên hãy ứng dụng kiến thức về lập trình hướng đối tượng để thực hiện các yêu cầu sau:

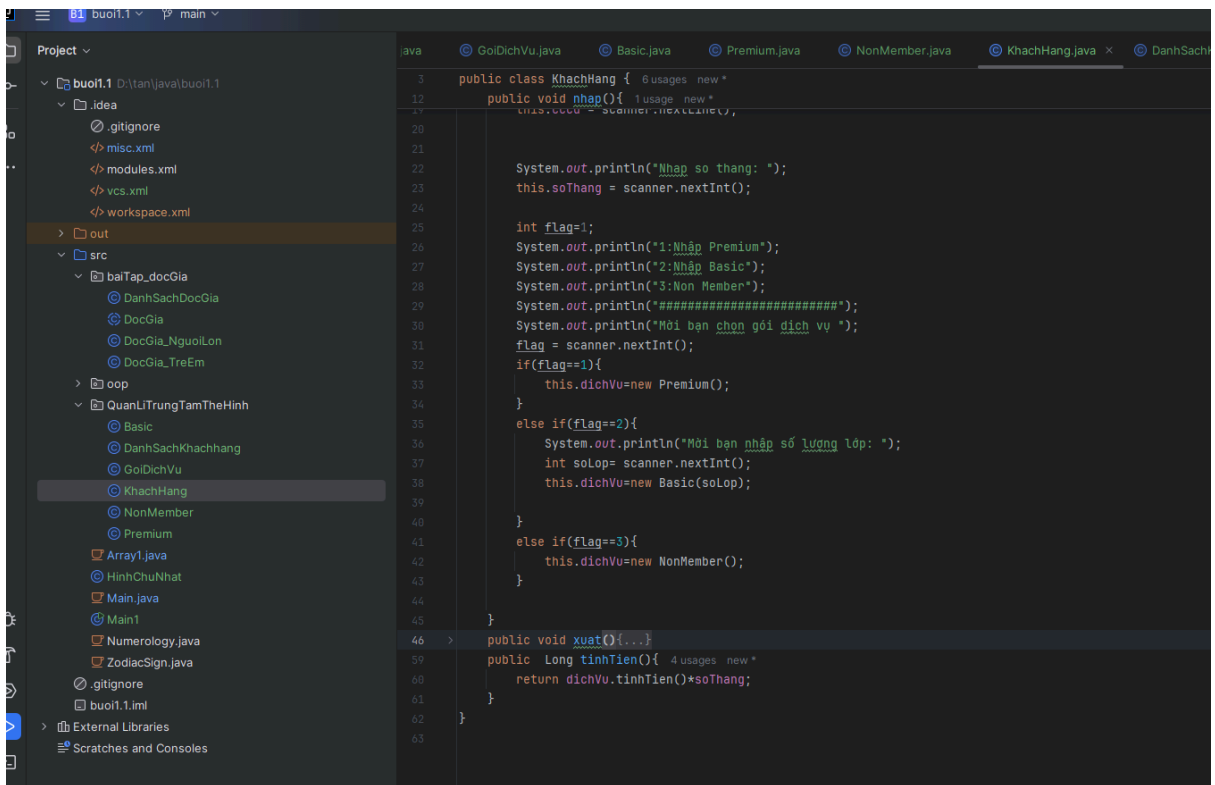
1. Xây dựng sơ đồ phân lớp kế thừa. (1 đ)
2. Cài đặt các lớp thích hợp. (2 đ)
3. Quản lý việc nhập xuất danh sách khách hàng. (1 đ)
4. Trung tâm cần thông tin của những khách hàng đã chi tiêu nhiều nhất để tiến hành tư vấn gói dịch vụ thích hợp cho họ. Hãy viết phương thức thực hiện chức năng này. (1 đ)



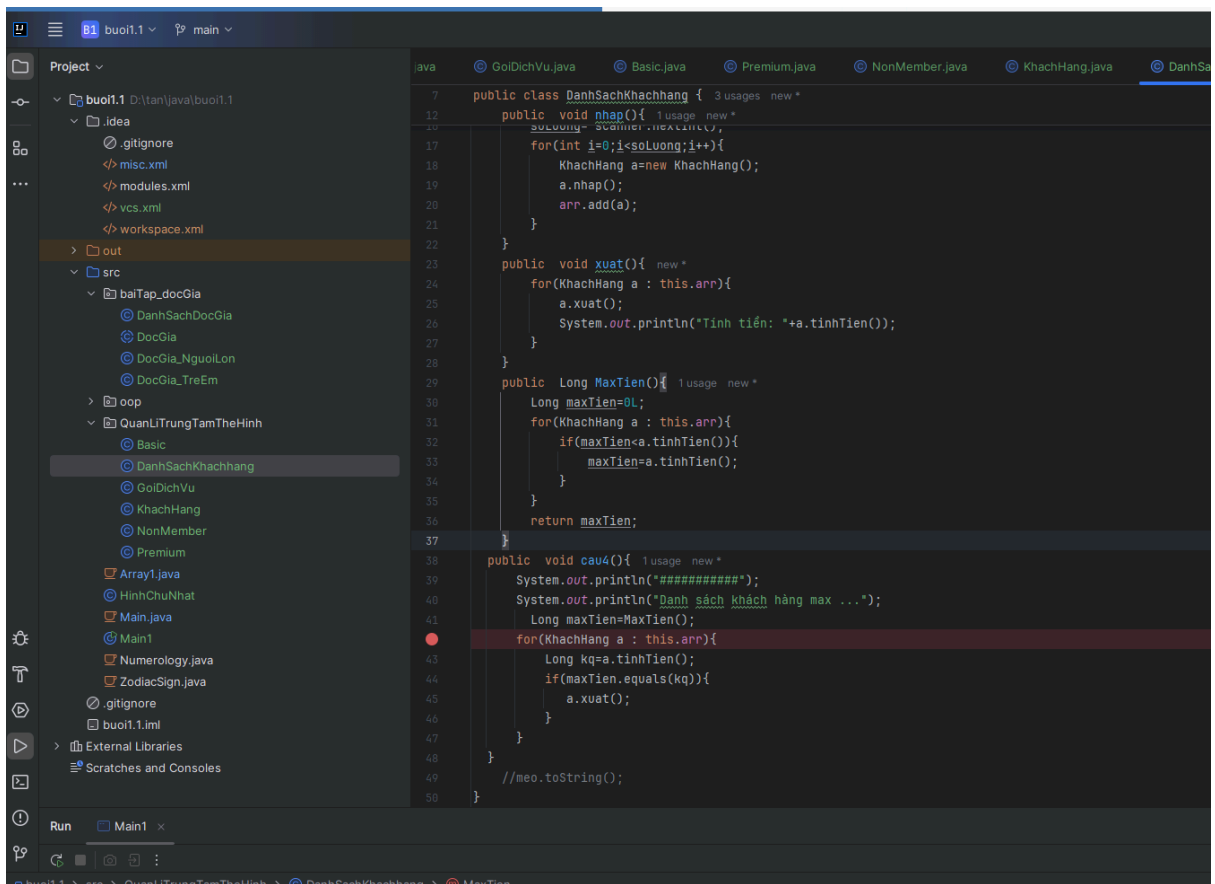
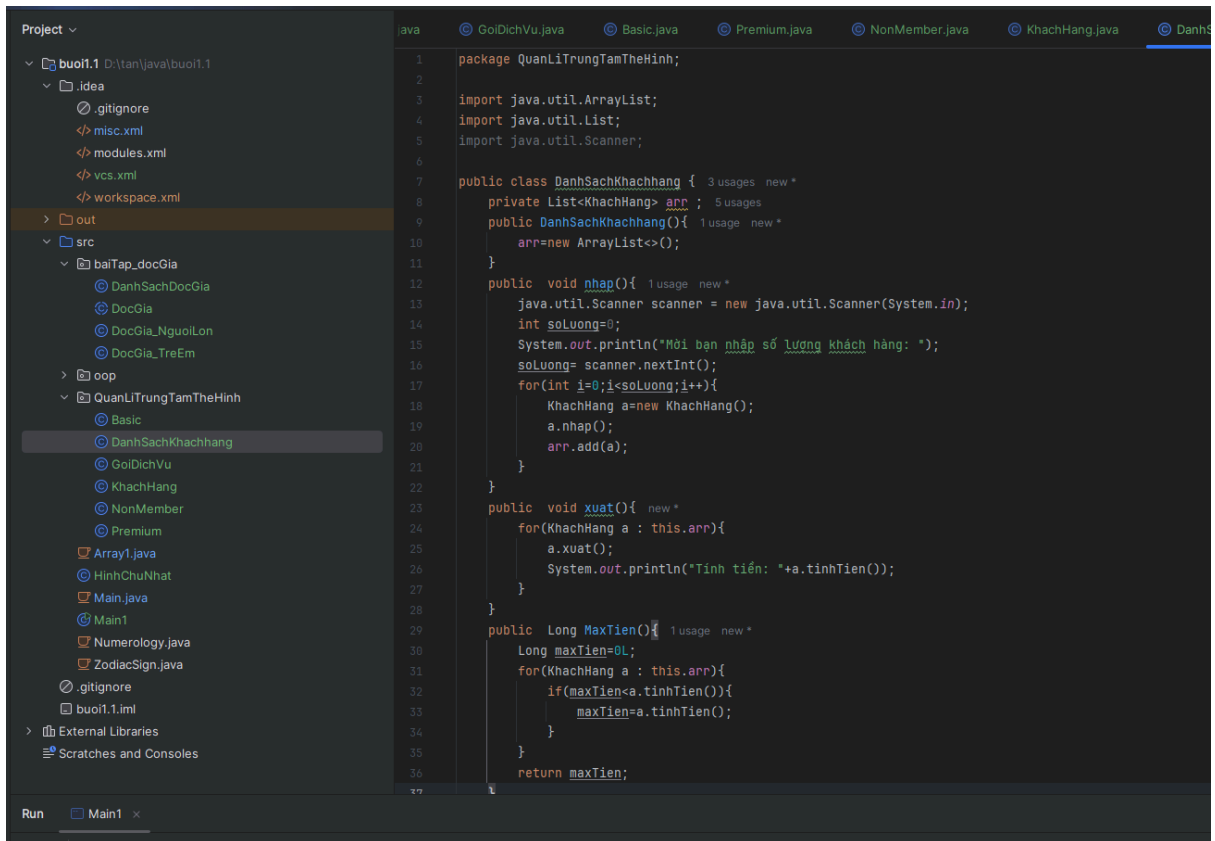




```
1 package QuanLiTrungTamTheHinh;
2
3 public class KhachHang { 6 usages new *
4     private String hoTen; 2 usages
5     private String cccd; 2 usages
6     private int soThang; 3 usages
7     private GoiDichVu dichVu; 7 usages
8
9     public KhachHang(){ 1 usage new *
10
11     }
12     public void nhap(){ 1 usage new *
13         java.util.Scanner scanner = new java.util.Scanner(System.in);
14         System.out.println("Nhập Tên khách hàng: ");
15         this.hoTen = scanner.nextLine();
16
17         System.out.println("Nhập cccd: ");
18         this.cccd = scanner.nextLine();
19
20         System.out.println("Nhập số tháng: ");
21         this.soThang = scanner.nextInt();
22
23         int flag=1;
24         System.out.println("1:Nhập Premium");
25         System.out.println("2:Nhập Basic");
26         System.out.println("3:Non Member");
27         System.out.println("*****");
28         System.out.println("Mời bạn chọn gói dịch vụ ");
29         flag = scanner.nextInt();
30         if(flag==1){
31             this.dichVu=new Premium();
32         }
33         else if(flag==2){
34             System.out.println("Mời bạn nhập số lượng lớp: ");
35             int soLop= scanner.nextInt();
36
37         }
```



```
3 public class KhachHang { 6 usages new *
12     public void nhap(){ 1 usage new *
13         java.util.Scanner scanner = new java.util.Scanner(System.in);
14         System.out.println("Nhập Tên khách hàng: ");
15         this.hoTen = scanner.nextLine();
16
17         System.out.println("Nhập số tháng: ");
18         this.soThang = scanner.nextInt();
19
20         int flag=1;
21         System.out.println("1:Nhập Premium");
22         System.out.println("2:Nhập Basic");
23         System.out.println("3:Non Member");
24         System.out.println("*****");
25         System.out.println("Mời bạn chọn gói dịch vụ ");
26         flag = scanner.nextInt();
27         if(flag==1){
28             this.dichVu=new Premium();
29         }
30         else if(flag==2){
31             System.out.println("Mời bạn nhập số lượng lớp: ");
32             int soLop= scanner.nextInt();
33             this.dichVu=new Basic(soLop);
34         }
35         else if(flag==3){
36             this.dichVu=new NonMember();
37         }
38     }
39
40     public void xuat(){...}
41
42     public Long tinhTien(){ 4 usages new *
43         return dichVu.tinhTien()*soThang;
44     }
45
46 }
```



Long.compare(long x, long y): So sánh hai giá trị kiểu long. Trả về 0 nếu chúng bằng nhau, một giá trị âm nếu x nhỏ hơn y, và một giá trị dương nếu x lớn hơn y.

Long.compareUnsigned(long x, long y): So sánh hai giá trị kiểu long theo cách không dấu (unsigned). Thích hợp khi làm việc với số nguyên không dấu.

Collection:

- Trong Java, có một số loại collection (tập hợp) cơ bản mà bạn có thể sử dụng để lưu trữ và quản lý dữ liệu

1.List

- **List** là một tập hợp có thứ tự và cho phép các phần tử trùng lặp. Các triển khai phổ biến của **List** bao gồm:
- **ArrayList**: Cung cấp hiệu suất tốt khi truy cập ngẫu nhiên và có khả năng mở rộng động.
- Các ví dụ:

Các ví dụ:

```

import java.util.ArrayList;
import java.util.Arrays;
import java.util.Collections;
import java.util.Comparator;

public class Main3 { new *
    public static void main(String[] args) { new *

        ArrayList<String> colors = new ArrayList<>();
        colors.add("Red");
        colors.add("Green");
        colors.add("Blue");

        // Truy cập phần tử tại vị trí chỉ định
        System.out.println(colors.get(1));

        // Thay đổi phần tử tại vị trí chỉ định
        colors.set(1, "Yellow");

        // Xóa phần tử theo giá trị
        colors.remove(o: "Red");
        System.out.println(colors);

        // Xóa phần tử theo chỉ số
        colors.remove(index: 1);
        System.out.println(colors);

        for (String color : colors) {
            System.out.println(color);
        }
        // System.out.println(colors);
        // Tạo một mảng
        String[] colorsArray = {"Red", "Green", "Blue"};

        // Tạo ArrayList từ mảng
        ArrayList<String> colorsList = new ArrayList<>(Arrays.asList(colorsArray));
        // sắp xếp mảng: cách thường
        Collections.sort(colorsList);
        // Sử dụng biểu thức lambda để sắp xếp danh sách theo thứ tự tăng dần
        Collections.sort(colorsList, (s1, s2) -> s1.compareTo(s2));
        System.out.println(colorsList); // Output: [Red, Green, Blue]
    }
}

```


© DanhsachKhachhang.java © DanhsachDocGia.java Main.java © Nguo

```

1  public class Point { 7 usages new *
2      private int x; 5 usages
3      private int y; 5 usages
4
5      // Constructor
6      public Point(int x, int y) { 3 usages new *
7          this.x = x;
8          this.y = y;
9      }
10     public Point(){ no usages new *
11
12     }
13     @ public Point(Point b){ no usages new *
14         this.x=b.x;
15         this.y=b.y;
16     }
17
18     // Getter cho tọa độ x
19     public int getX() { 1 usage new *
20         return x;
21     }
22
23     // Getter cho tọa độ y
24     public int getY() { 1 usage new *
25         return y;
26     }
27
28     // Override phương thức toString để dễ dàng in đối tượng
29     @Override new *
30     @ public String toString() {
31         return "(" + x + ", " + y + ")";
32     }
33 }
34
  
```

```

// Tạo ArrayList từ mảng
ArrayList<String> colorsList = new ArrayList<>(Arrays.asList(colorsArray));
//sắp xếp mảng: cách thường
Collections.sort(colorsList);
// Sử dụng biểu thức lambda để sắp xếp danh sách theo thứ tự tăng dần
Collections.sort(colorsList, (s1, s2) -> s1.compareTo(s2));
System.out.println(colorsList); // Output: [Red, Green, Blue]

ArrayList<Point> points = new ArrayList<>();
points.add(new Point(3, 7));
points.add(new Point(3, 6));
points.add(new Point(5, 2));
Comparator<Point> x_y_compare = new Comparator<Point>() {
    @Override
    public int compare(Point o1, Point o2) {
        return Integer.compare(o1.getX(), o2.getX());
    }
    public int compare(Point p1, Point p2) {
        int tmp=Integer.compare(p1.getX(), p2.getX());
        if(tmp==0){
            return Integer.compare(p1.getY(), p2.getY());
        }
        return tmp;
    }
    public int compare(Point p1, Point p2) {
        return Integer.compare(p1.getX()+p1.getY(), p2.getX()+p2.getY());
    }
};
Collections.sort(points, x_y_compare);
for (Point p : points) {
    System.out.println(p);
}

```

1. **Định nghĩa:** Biểu thức lambda trong Java (còn được gọi là lambda expression) là một tính năng được giới thiệu từ Java 8, giúp bạn viết mã ngắn gọn hơn và dễ đọc hơn khi làm việc với các đối tượng như `Runnable`, `Comparator`, `Function`, `Predicate`, và nhiều loại `functional interface` khác.
2. **Ưu điểm:** Biểu thức lambda mang lại nhiều lợi ích như giảm độ dài mã, cải thiện tính đọc hiểu, hỗ trợ lập trình hàm, tăng tính tái sử dụng và hiệu suất. Sử dụng lambda expressions là một cách hiệu quả để viết mã sạch hơn và dễ bảo trì hơn trong các ứng dụng Java hiện đại.
3. **Nội dung:** các biểu thức lambda thường được sử dụng với các `functional interfaces` để đơn giản hóa mã và cải thiện khả

năng đọc. Các `functional interfaces` phổ biến bao gồm `Predicate`, `Function`, `Consumer`, và `Supplier`.

a. `Predicate`:

+Sử dụng `Predicate` khi bạn cần thực hiện một kiểm tra hoặc điều kiện và trả về giá trị boolean.

+**Phương Thức Chính:** `boolean test(T t);`

+ **VD Cú Pháp Lambda:** `Predicate<String> isEmpty = s -> s.isEmpty();`

b. `Function`:

+Là một `functional interface` nhận một đối tượng đầu vào và trả về một kết quả. Có thể dùng để biến đổi hoặc xử lý dữ liệu.

+**Phương Thức Chính:** `R apply(T t);`

+**VD Cú Pháp Lambda:** `Function<String, Integer> stringLength = s -> s.length();`

+Sử dụng `Function` khi bạn cần chuyển đổi hoặc xử lý dữ liệu và nhận được kết quả là một đối tượng khác.

c. `Consumer`

+ `Consumer` là một `functional interface` nhận một đối tượng đầu vào và thực hiện một hành động trên đối tượng đó mà không trả về giá trị.

+Phương Thức Chính: `void accept(T t);`

+VD Cú Pháp Lambda: `Consumer<String>
printUpperCase = s ->
System.out.println(s.toUpperCase());`

+Sử dụng `Consumer` khi bạn cần thực hiện một hành động trên dữ liệu mà không cần trả về giá trị.

d. Stream

- + **Stream** không phải là một `functional interface` mà là một API trong Java 8 để làm việc với tập hợp dữ liệu theo cách hàm (functional style). Streams cho phép bạn thực hiện các phép toán trên dữ liệu, chẳng hạn như lọc, ánh xạ, và sắp xếp.
- + **Phương Thức Chính:** **Stream** không có một phương thức chính duy nhất; thay vào đó, nó cung cấp nhiều phương thức như `filter`, `map`, `reduce`, v.v.
- + **Cú Pháp Lambda:** Sử dụng các biểu thức lambda trong các phép toán của stream.
- + Sử dụng **Stream** khi bạn muốn xử lý dữ liệu tập hợp theo cách hàm, cho phép thực hiện các phép toán phức tạp trên dữ liệu một cách dễ dàng và hiệu quả.
- + Các hàm:

Tạo Stream: *Bạn có thể tạo một stream từ các tập hợp dữ liệu như danh sách, mảng, hoặc tạo stream từ các giá trị cụ thể.*

Lọc: *Sử dụng `filter` để giữ lại các phần tử thỏa mãn điều kiện.*

Ánh Xạ: *Sử dụng `map` để chuyển đổi từng phần tử trong stream.*

Sắp Xếp: *Sử dụng `sorted` để sắp xếp các phần tử trong stream.*

Tính Toán: Sử dụng các phương thức như *sum*, *average*, *max*, *min* để thực hiện các phép toán tổng hợp.

Kết Hợp Phép Toán: Sử dụng nhiều phép toán kết hợp để xử lý dữ liệu theo cách mong muốn.

FlatMap: Kết hợp các danh sách con thành một danh sách duy nhất.

Giảm: Sử dụng *reduce* để thực hiện các phép toán tổng hợp hoặc tính toán phức tạp.

VD: Biểu thức lambda để tính bình phương của một số
Function<Integer, Integer> square = x -> x * x; // Sử dụng biểu thức lambda

```
System.out.println(square.apply(5)); // Output: 25
```

Ví dụ 2:

// Biểu thức lambda để kiểm tra xem một số có phải là số chẵn không
Predicate<Integer> isEven = num -> { return num % 2 == 0; };

```
System.out.println(isEven.test(4)); // Output: true  
System.out.println(isEven.test(5)); // Output: false
```

```

System.out.println("#####");
ArrayList<Point> points = new ArrayList<>();
points.add(new Point(x: 3, y: 5));
points.add(new Point(x: 2, y: 6));
points.add(new Point(x: 5, y: 2));
points.add(new Point(x: 6, y: 8));

System.out.println("#####");
Predicate<Point> SumPoint = point -> point.getX() + point.getY() > 7;
Predicate<Point> SumPoint2 = point -> {
    if (point.getX() > 5) {
        return point.getX() + point.getY() > 7;
    } else {
        return point.getX() + point.getY() > 6;
    }
};
for (Point point : points) {
    if (SumPoint2.test(point)) {
        System.out.println(point.toString());
    }
}
Function<Point, Integer> sumPoint3 = p -> p.getY() + p.getX();
Function<Point, Integer> sumPoint4 = p -> {
    return p.getX() > p.getY() ? p.getY() - p.getX() : p.getY() + p.getX();
};
for (Point point : points) {
    System.out.println(sumPoint4.apply(point));
}
Consumer<Point> printMessage = p -> {
    if (p.getX() > p.getY()) {
        System.out.println(p.getY() - p.getX());
    }
}

```

```

public class Main4 { new *
    public static void main(String[] args) { new *

        Consumer<Point> printMessage = p -> {
            if (p.getX() > p.getY()) {
                System.out.println(p.getY() - p.getX());
            }
            else{
                System.out.println(p.getY() + p.getX());
            }
        };

        System.out.println("#####");
        points.forEach(printMessage);
        System.out.println("#####");
        for (Point point : points) {
            printMessage.accept(point);
        }

        System.out.println("#####");
        points.stream()
            .filter(p -> p.getX() > 3)
            .forEach(System.out::println);
        System.out.println("#####");
        for(Point a:points){
            if(a.getX()>3){
                System.out.println(a.toString());
            }
        }

        System.out.println("#####");
        Predicate<Point> pre_x = point -> point.getX() > 3;
        for (Point point : points) {
            if (pre_x.test(point)) {
                System.out.println(point.toString());
            }
        }

        System.out.println("#####");
        Consumer<Point> consum_x = p -> {

```

```
© DanhSachKhachhang.java © DanhSachDocGia.java ☞ Main1.java ⓘ Ngnoi.java
7 public class Main4 { new *
8     public static void main(String[] args) { new *
9
10         System.out.println("#####");
11         points.stream()
12             .filter(p -> p.getX() > 3)
13             .forEach(System.out::println);
14         System.out.println("#####");
15         for(Point a:points){
16             if(a.getX()>3){
17                 System.out.println(a.toString());
18             }
19         }
20         System.out.println("#####");
21         Predicate<Point> pre_x = point -> point.getX() > 3;
22         for (Point point : points) {
23             if (pre_x.test(point)) {
24                 System.out.println(point.toString());
25             }
26         }
27         System.out.println("#####");
28         Consumer<Point> consum_x = p -> {
29             if (p.getX() > 3) {
30                 System.out.println(p.toString());
31             }
32         };
33         points.forEach(consum_x);
34     }
35 }
36 }
```

Việc chuyển đổi từ `Stream<Integer>` thành `IntStream` giúp bạn:

- **Tăng hiệu suất:** Vì làm việc với kiểu dữ liệu nguyên thủy (`int`) hiệu quả hơn so với đối tượng bao bọc (`Integer`).
- **Sử dụng các phương thức số học tích hợp sẵn:** Như `sum()`, `average()`, và các phép toán khác mà `IntStream` cung cấp.

- **Giảm tiêu tốn bộ nhớ:** Bằng cách tránh việc lưu trữ các đối tượng bao bọc không cần thiết.

```
System.out.println("#####");
List<String> names = Arrays.asList("John", "Alice", "Bob", "Anna");

// Lọc các tên bắt đầu bằng 'A'
names.stream()
    .filter(name -> name.startsWith("A"))
    .forEach(System.out::println); // Output: Alice, Anna

//
//
//
names.stream()
    .filter(name -> name.startsWith("A"))
    .forEach(item -> System.out.println(item)); // Output: Alice, Anna

List<String> filteredNames = names.stream()
    .filter(name -> name.startsWith("A"))
    .map(String::toUpperCase)
    .sorted()
    .collect(Collectors.toList());

System.out.println(filteredNames); // Output: [ALICE, ANNA]

// Ánh xạ các tên thành chữ hoa
names.stream()
    .map(String::toUpperCase)
    .forEach(System.out::println); // Output: JOHN, ALICE, BOB
```

```

public static void main(String[] args) {
    System.out.println("Hello Java!"); // Output: Hello, World!

    // Ánh xạ các tên thành chữ hoa
    names.stream()
        .map(String::toUpperCase)
        .forEach(System.out::println); // Output: JOHN, ALICE, BOB

    names.stream()
        .sorted()
        .forEach(System.out::println); // Output: Alice, Bob, John

    // Sắp xếp theo thứ tự giảm dần
    names.stream()
        .sorted(Comparator.reverseOrder())
        .forEach(System.out::println); // Output: John, Bob, Alice

    List<Integer> numbers = Arrays.asList(1, 2, 3, 4, 5);

    // Tính tổng
    int sum = numbers.stream() Stream<Integer>
        .mapToInt(Integer::intValue) IntStream
        .sum();
    System.out.println("Sum: " + sum); // Output: Sum: 15

    // stream<Integer> => int
    // Tính trung bình
    double average = numbers.stream() Stream<Integer>
        .mapToInt(Integer::intValue) IntStream
        .average() OptionalDouble
        .orElse(0);
    System.out.println("Average: " + average); // Output: Average: 3.0

    // Tìm giá trị lớn nhất
    OptionalInt max = numbers.stream() Stream<Integer>
        .mapToInt(Integer::intValue) IntStream
        .max();
    System.out.println("Max: " + (max.isPresent() ? max.getAsInt() : "Not present")); // Output: Max: 5
}

```

```

// Tìm giá trị lớn nhất
OptionalInt max = numbers.stream() Stream<Integer>
    .mapToInt(Integer::intValue) IntStream
    .max();
System.out.println("Max: " + (max.isPresent() ? max.getAsInt() : "Not present")); // Output: Max: 5

// Tìm giá trị nhỏ nhất
OptionalInt min = numbers.stream() Stream<Integer>
    .mapToInt(Integer::intValue) IntStream
    .min();
System.out.println("Min: " + (min.isPresent() ? min.getAsInt() : "Not present")); // Output: Min: 1

// Sử dụng reduce để tính tổng
int sum1 = numbers.stream()
    .reduce( identity: 0, Integer::sum);
System.out.println("Sum using reduce: " + sum1); // Output: Sum using reduce: 15

// Sử dụng reduce để tìm tích
int product = numbers.stream()
    .reduce( identity: 1, (a, b) -> a * b);
System.out.println("Product using reduce: " + product); // Output: Product using reduce: 120

List<List<String>> listOfLists = Arrays.asList(
    Arrays.asList("a", "b", "c"),
    Arrays.asList("d", "e", "f"),
    Arrays.asList("g", "h", "i")
);

// Sử dụng flatMap để kết hợp các danh sách con thành một danh sách
List<String> combinedList = listOfLists.stream() Stream<List<...>>
    .flatMap(List::stream) Stream<String>
    .collect(Collectors.toList());

System.out.println(combinedList); // Output: [a, b, c, d, e, f, g, h, i]
}

```

Ví dụ kiểu point

- Viết 4 từ tiếng anh: kế thừa, đa hình, trừa tượng, đóng gói

Lê Khoa, Hoàng Hảo, ngọc Hải, Ngọc Nguyên

- Tính tổng các x,y của danh sách:

+ngọc đạt, Văn Hoàng, Hoàng Minh

-3 bạn giải đúng: Khoa, Hoàng, Sỹ

```
//  
int vd = points.stream()  
    .reduce( identity: 0, (acc, p) -> acc + p.getX() + p.getY(), Integer::sum);  
  
int vd2 = points.stream()  
    .map(p -> p.getX() + p.getY()) // Chuyển đổi từng Point thành tổng x + y  
    .reduce( identity: 0, (acc, value) -> acc + value); // Tính tổng các giá trị
```

```

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

public class Stream1 { new *
    public static void test_stream_point(){ 1 usage new *
        List<Point> points = new ArrayList<>();
        points.add(new Point(x: 7, y: 2));
        points.add(new Point(x: 3, y: 4));
        points.add(new Point(x: 2, y: 6));
        points.add(new Point(x: 7, y: 1));
        int totalSum = points.stream() Stream<Point>
            .flatMapToInt(point -> Arrays.stream(new int[]{point.getX(), point.getY()})) IntStream
            .sum();
        //Tinh tổng các hoành độ
        int sumX = points.stream()
            .mapToInt(Point::getX) // Chuyển đổi thành IntStream dựa trên giá trị x
            .sum(); // Tính tổng

        //Tinh tổng các hoành độ >5
        int sumX2 = points.stream()
            .filter(p->p.getX()>5)
            .mapToInt(Point::getX) // Chuyển đổi thành IntStream dựa trên giá trị x
            .sum();

        //tinh tổng toàn bộ x^2+1-y đối với các hoành độ >5
        int sumX3 = points.stream()
            .filter(p->p.getX()>5)
            .mapToInt(p->{
                return p.getX()*p.getX()+1-p.getY();
            }) // Chuyển đổi thành IntStream dựa trên giá trị x
            .sum();

        //In ra từng kết quả x^2+1-y đối với x>4
        points.stream()
            .filter(p->p.getX()>4)
            .mapToInt(p->{
                return p.getX()*p.getX()+1-p.getY();
            })
    }
}

```

```

public static void test_stream_point(){ 1 usage new *
    //In ra từng kết quả  $x^2+1-y$  đối với  $x>4$ 
    points.stream()
        .filter(p->p.getX()>4)
        .mapToInt(p->{
            return p.getX()*p.getX()+1-p.getY();
        }) // Chuyển đổi thành IntStream dựa trên giá trị x
        .forEach(res ->System.out.println(res));
    //kq= 48, kq=49

    // tính tích toàn bộ  $x^2+1-y$  đối với các hoành độ >1
    long total5=points.stream()
        .filter(p->p.getX()>1)
        .mapToLong(p->{
            return p.getX()*p.getX()+1-p.getY();
        }) // Chuyển đổi thành IntStream dựa trên giá trị x
        .reduce(Identity: 1L, (kq, item)->kq*item);
    System.out.println("total5 = "+total5);

    // Tính tích các giá trị của  $1 / (x^2 + y)$  cho tất cả các Point
    double totalProduct = points.stream() Stream<Point>
        .mapToDouble(point -> {
            int x = point.getX();
            int y = point.getY();
            double tmp = x * x + y; // Tính  $x^2 + y$ 
            return 1.0 / tmp; // Tính  $1 / (x^2 + y)$ 
        }) DoubleStream
        .reduce(Identity: 1.0, (res, item) -> res * item); // Tính tích toàn bộ các giá trị

    // In kết quả
    System.out.println("Tích toàn bộ giá trị của  $1 / (x^2 + y)$ : " + totalProduct);

    // in ra danh sách  $x+y>7$ 
    points.stream()
        .filter(p->(p.getX()+p.getY())>7))
        .forEach(System.out::println);

```

```

// Tính tích các giá trị của 1 / (x^2 + y) cho tất cả các Point
double totalProduct = points.stream() Stream<Point>
    .mapToDouble(point -> {
        int x = point.getX();
        int y = point.getY();
        double tmp = x * x + y; // Tính x^2 + y
        return 1.0 / tmp; // Tính 1 / (x^2 + y)
    }) DoubleStream
    .reduce(identity: 1.0, (res, item) -> res * item); // Tính tích toàn bộ các giá trị

// In kết quả
System.out.println("Tích toàn bộ giá trị của 1 / (x^2 + y): " + totalProduct);

// in ra danh sách x+y>7
points.stream()
    .filter(p->(p.getX()+p.getY())>7)
    .forEach(System.out::println);

System.out.println("#####");
// in ra danh sách x> 5 và y>1
points.stream()
    .filter(p->(p.getX()>5 && p.getY()>1))
    .forEach(System.out::println);

// In kết quả
System.out.println("Tổng tất cả các giá trị: " + totalSum);
System.out.println("Tổng tất cả các giá trị x: " + sumX);
System.out.println("Tổng tất cả các giá trị x2: " + sumX2);

}

public static void main(String[] args){ new *
    test_stream_point();
}

}

```