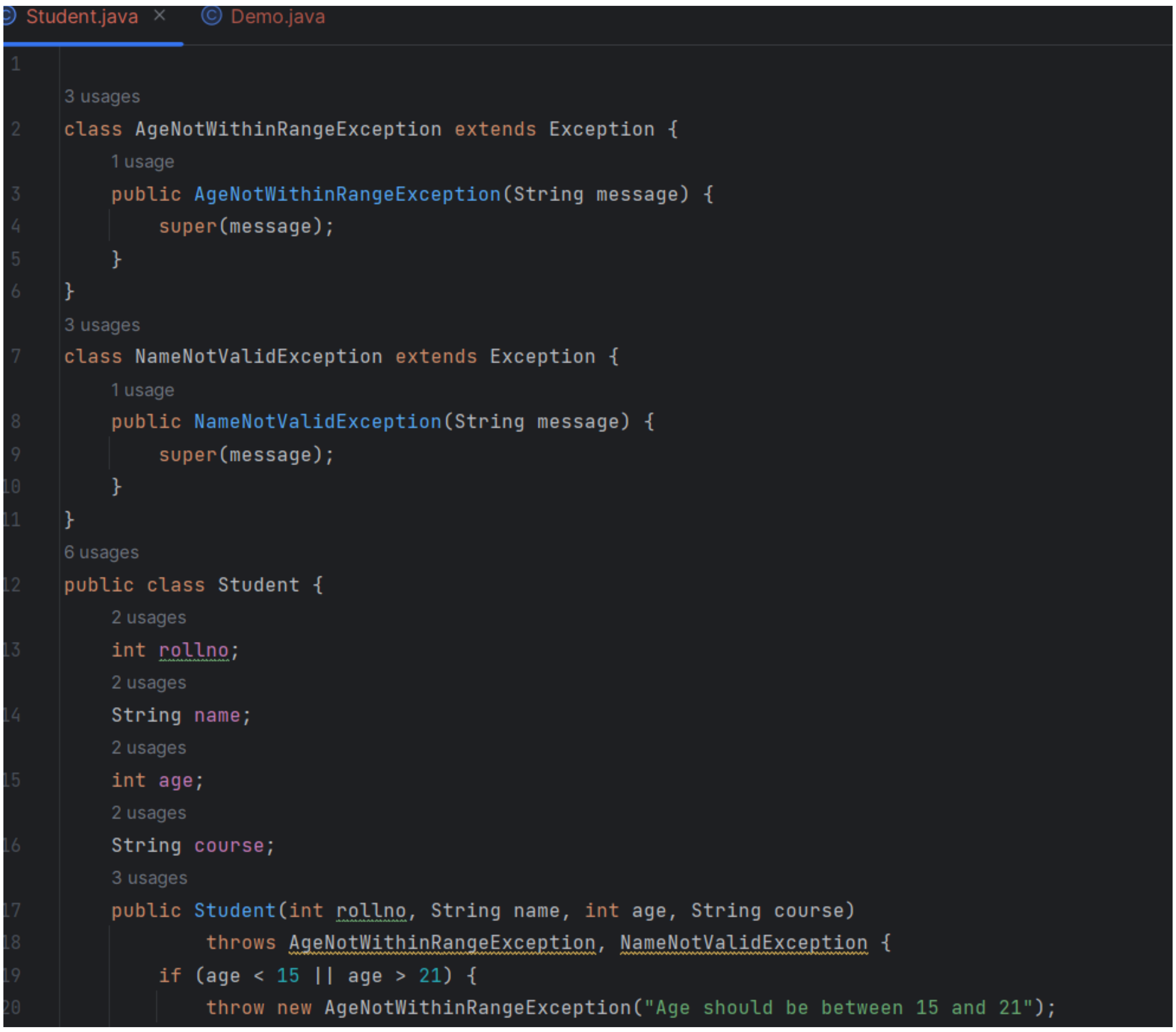
Task 4 Solutions

Q1. Ramesh is developing a student management system for a university. In this system, you have a Student class to represent student information. You are asked to help Ramesh to handle exception which can be occurred into program according to following Scenarios:

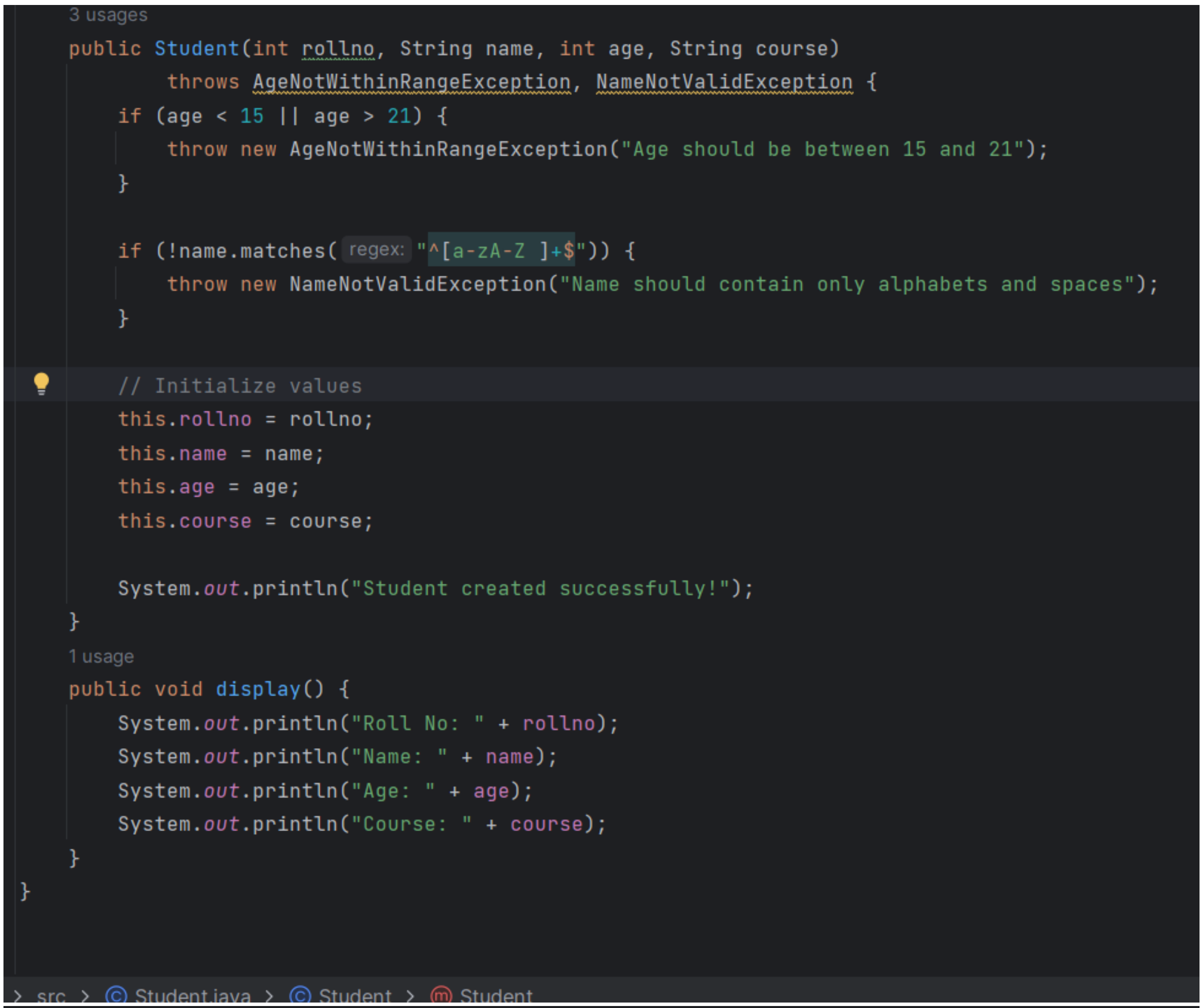
class Student with attributes roll no, name, age and course. Initialize values through parameterized constructors.

If the age of the student is not between 15 and 21 then generate a user-defined exception "AgeNotWithinRangeException".

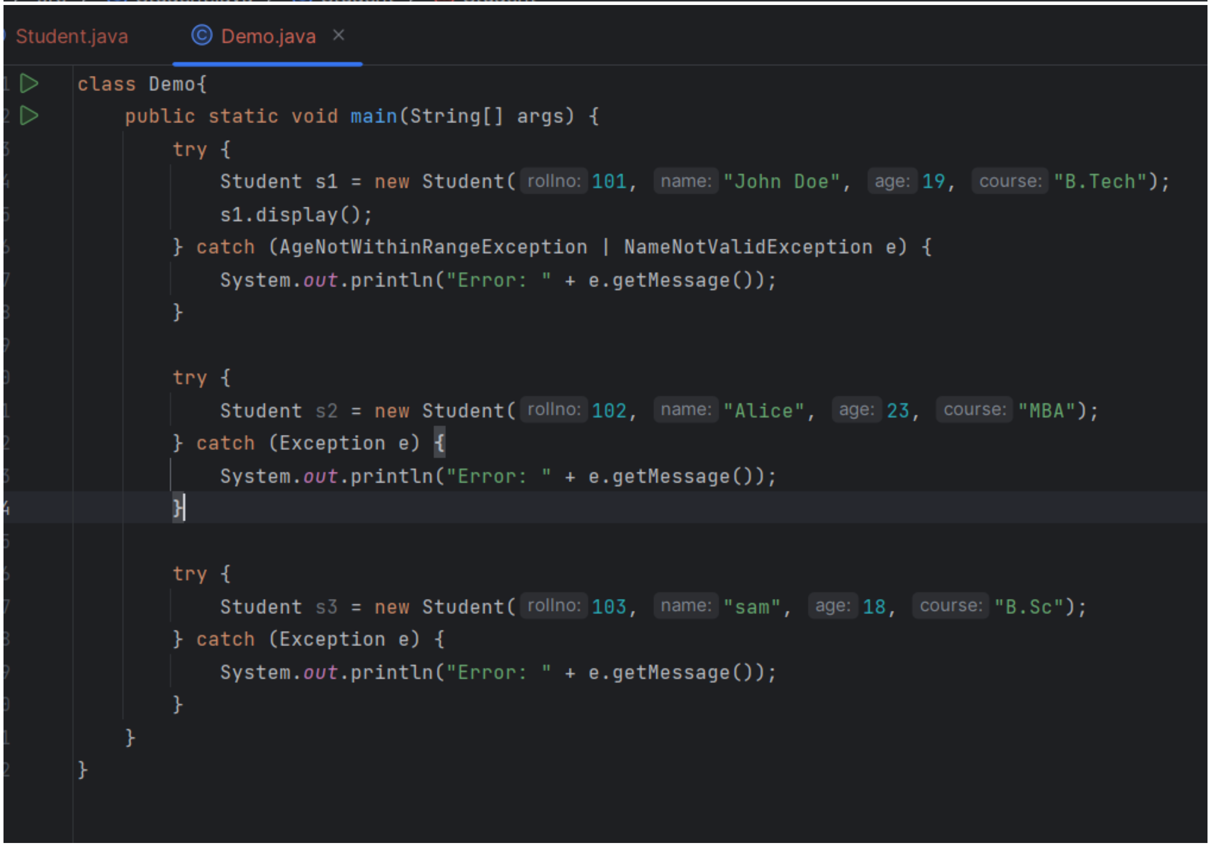
If a name contains numbers or special symbols, raise exception "NameNot ValidException". Define the two exception classes



```
3 usages
class AgeNotWithinRangeException extends Exception {
    1 usage
    public AgeNotWithinRangeException(String message) {
        super(message);
    }
}
3 usages
class NameNotValidException extends Exception {
    1 usage
    public NameNotValidException(String message) {
        super(message);
    }
}
6 usages
public class Student {
    2 usages
    int rollno;
    2 usages
    String name;
    2 usages
    int age;
    2 usages
    String course;
    3 usages
    public Student(int rollno, String name, int age, String course)
            throws AgeNotWithinRangeException, NameNotValidException {
        if (age < 15 || age > 21) {
            throw new AgeNotWithinRangeException("Age should be between 15 and 21");
```

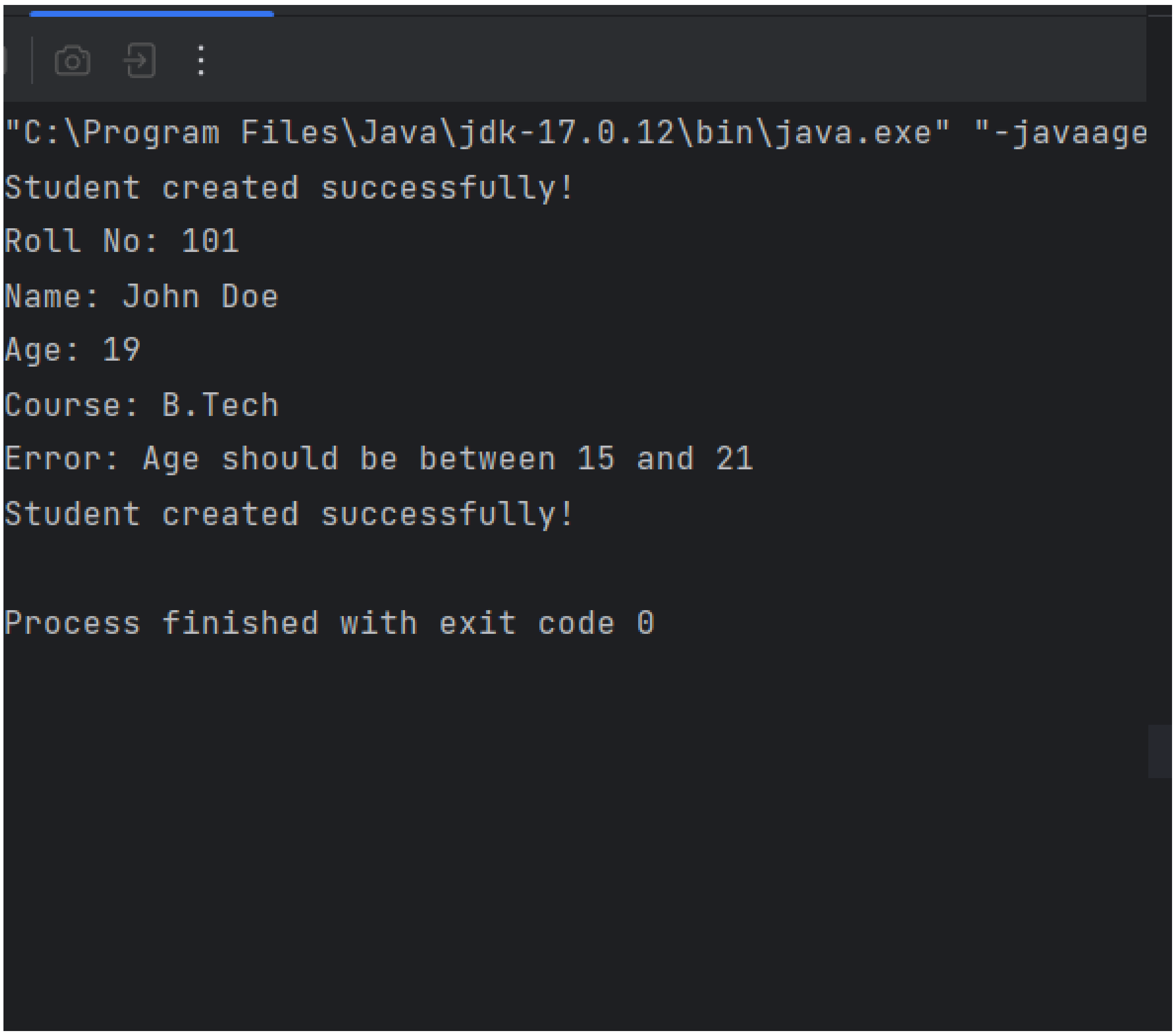


```java
    3 usages
    public Student(int rollno, String name, int age, String course)
            throws AgeNotWithinRangeException, NameNotValidException {
        if (age < 15 || age > 21) {
            throw new AgeNotWithinRangeException("Age should be between 15 and 21");
        }

        if (!name.matches( regex: "^[a-zA-Z ]+$")) {
            throw new NameNotValidException("Name should contain only alphabets and spaces");
        }

        // Initialize values
        this.rollno = rollno;
        this.name = name;
        this.age = age;
        this.course = course;

        System.out.println("Student created successfully!");
    }
    1 usage
    public void display() {
        System.out.println("Roll No: " + rollno);
        System.out.println("Name: " + name);
        System.out.println("Age: " + age);
        System.out.println("Course: " + course);
    }
}
```



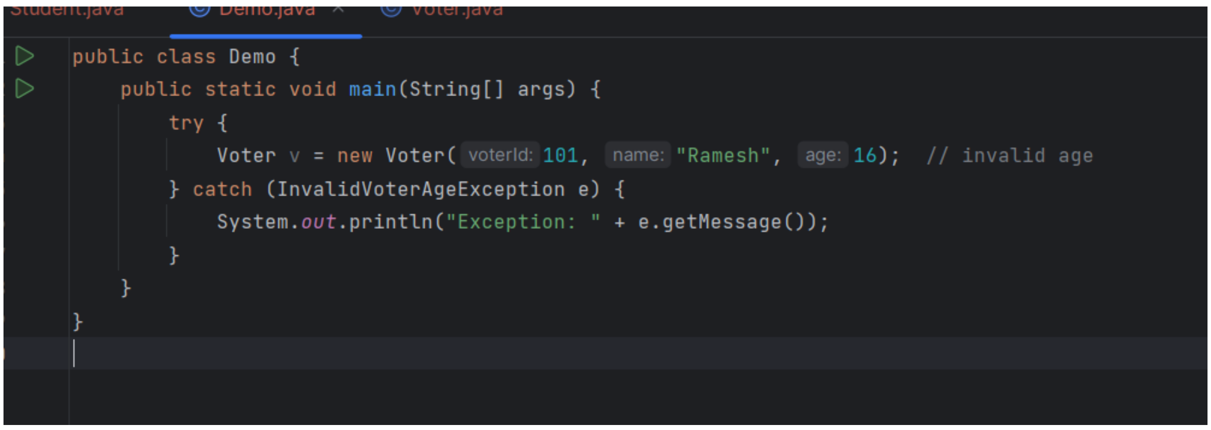
```java
class Demo{
    public static void main(String[] args) {
        try {
            Student s1 = new Student( rollno: 101,  name: "John Doe",  age: 19,  course: "B.Tech");
            s1.display();
        } catch (AgeNotWithinRangeException | NameNotValidException e) {
            System.out.println("Error: " + e.getMessage());
        }

        try {
            Student s2 = new Student( rollno: 102,  name: "Alice",  age: 23,  course: "MBA");
        } catch (Exception e) {
            System.out.println("Error: " + e.getMessage());
        }

        try {
            Student s3 = new Student( rollno: 103,  name: "sam",  age: 18,  course: "B.Sc");
        } catch (Exception e) {
            System.out.println("Error: " + e.getMessage());
        }
    }
}
```

Output:

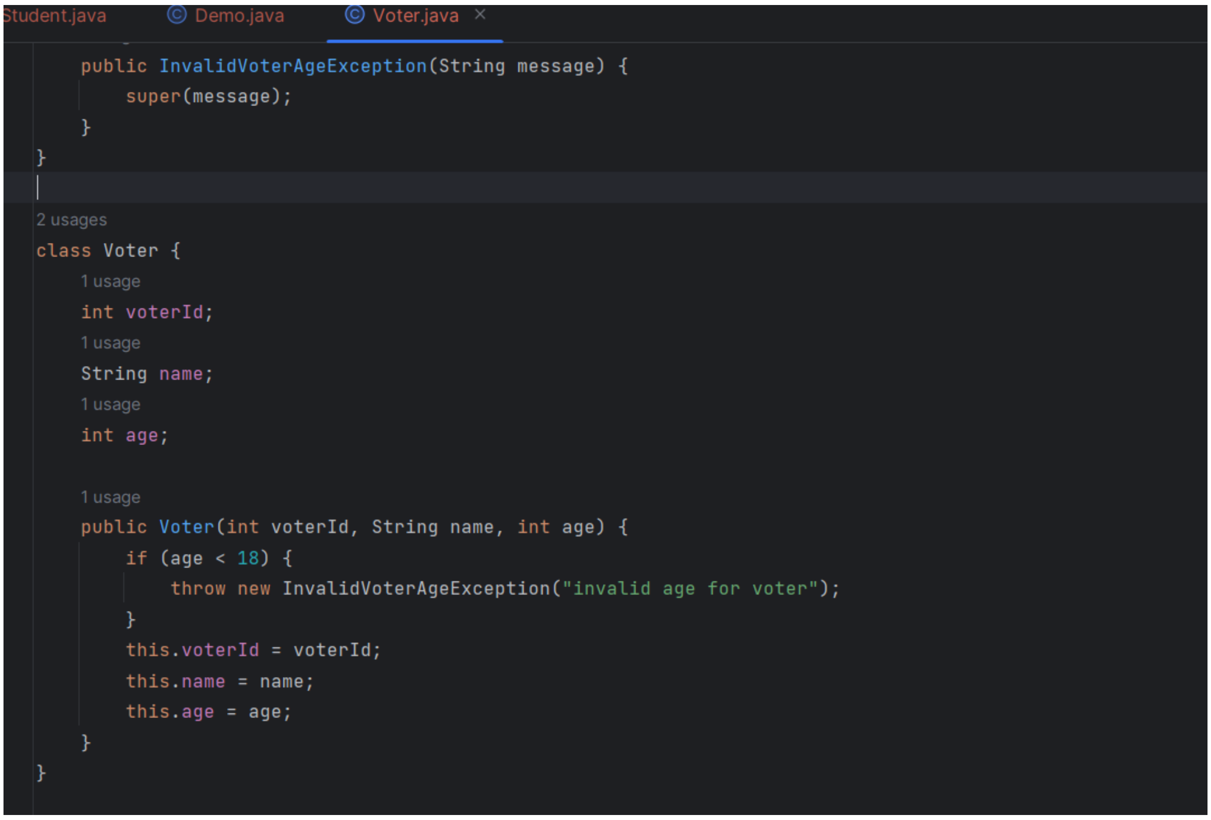


```
"C:\Program Files\Java\jdk-17.0.12\bin\java.exe" "-javaage
Student created successfully!
Roll No: 101
Name: John Doe
Age: 19
Course: B.Tech
Error: Age should be between 15 and 21
Student created successfully!

Process finished with exit code 0
```

Q2. Create a class Voter (voterId, name, age) with parameterized constructor. The parameterized constructor should throw a checked/Unchecked exception if age is less than 18. The message of exception is "invalid age for voter "

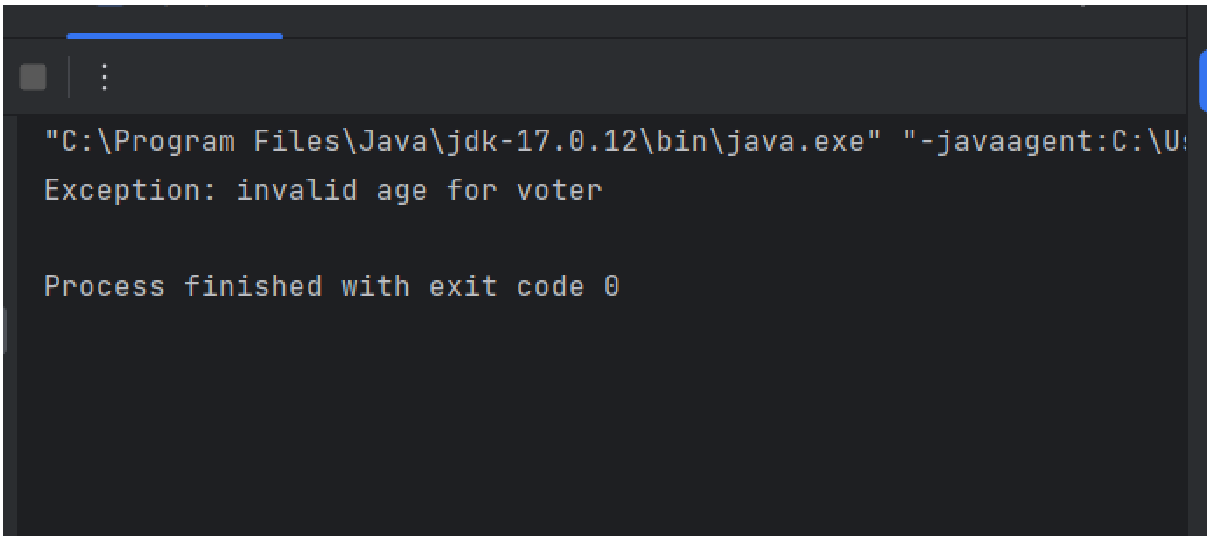


```java
public class Demo {
    public static void main(String[] args) {
        try {
            Voter v = new Voter( voterId: 101,  name: "Ramesh",  age: 16);  // invalid age
        } catch (InvalidVoterAgeException e) {
            System.out.println("Exception: " + e.getMessage());
        }
    }
}
```



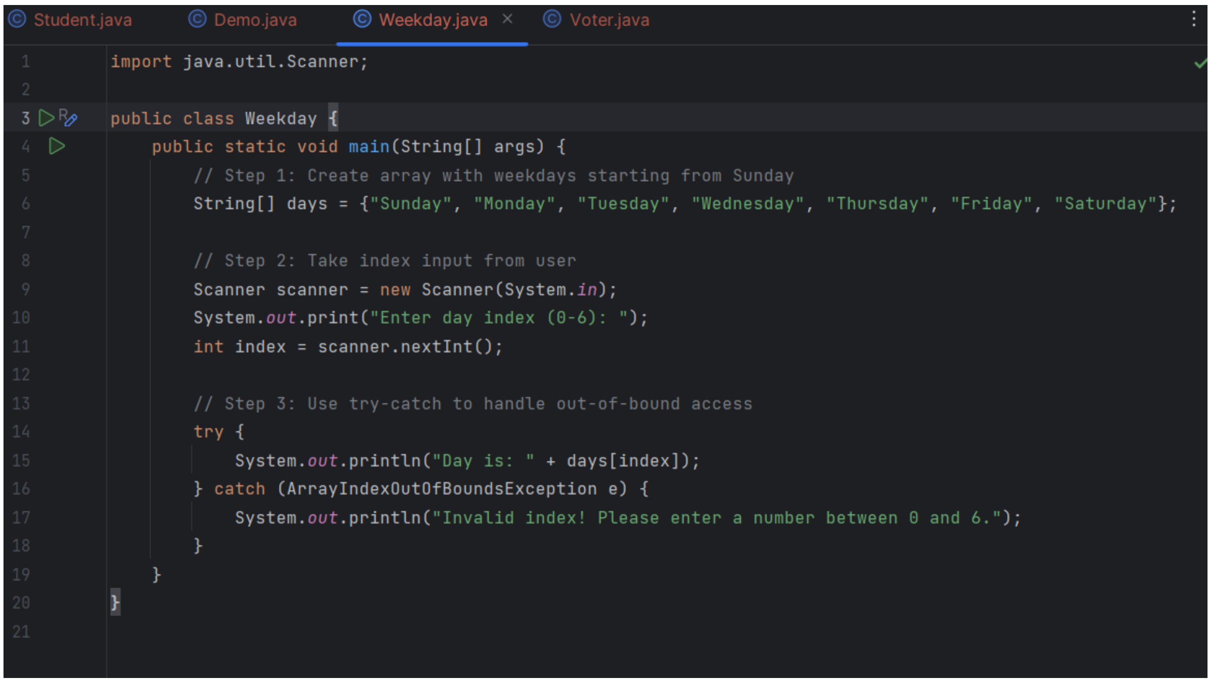
```java
    public InvalidVoterAgeException(String message) {
        super(message);
    }
}

2 usages
class Voter {
    1 usage
    int voterId;
    1 usage
    String name;
    1 usage
    int age;

    1 usage
    public Voter(int voterId, String name, int age) {
        if (age < 18) {
            throw new InvalidVoterAgeException("invalid age for voter");
        }
        this.voterId = voterId;
        this.name = name;
        this.age = age;
    }
}
```

Output:



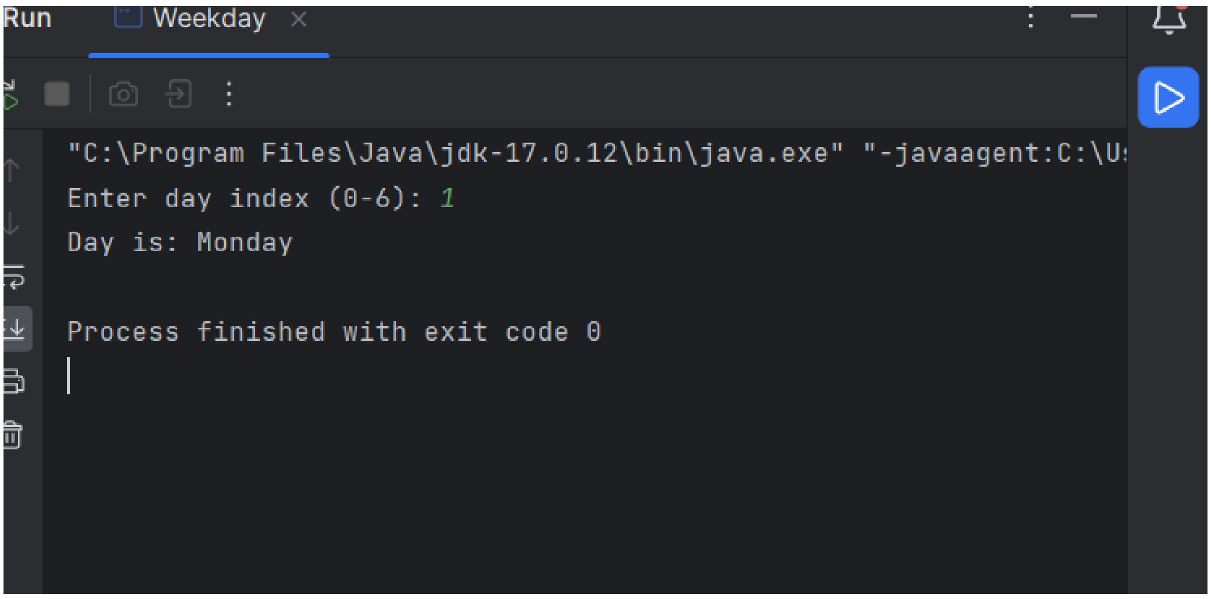
```
"C:\Program Files\Java\jdk-17.0.12\bin\java.exe" "-javaagent:C:\U
Exception: invalid age for voter

Process finished with exit code 0
```

Q3. Store name of weekdays in an array (starting from "Sunday" at 0 index). Ask day position from user and print day name. Handle array index out of bound exception and give proper message if user enters day index outside range (0-6).



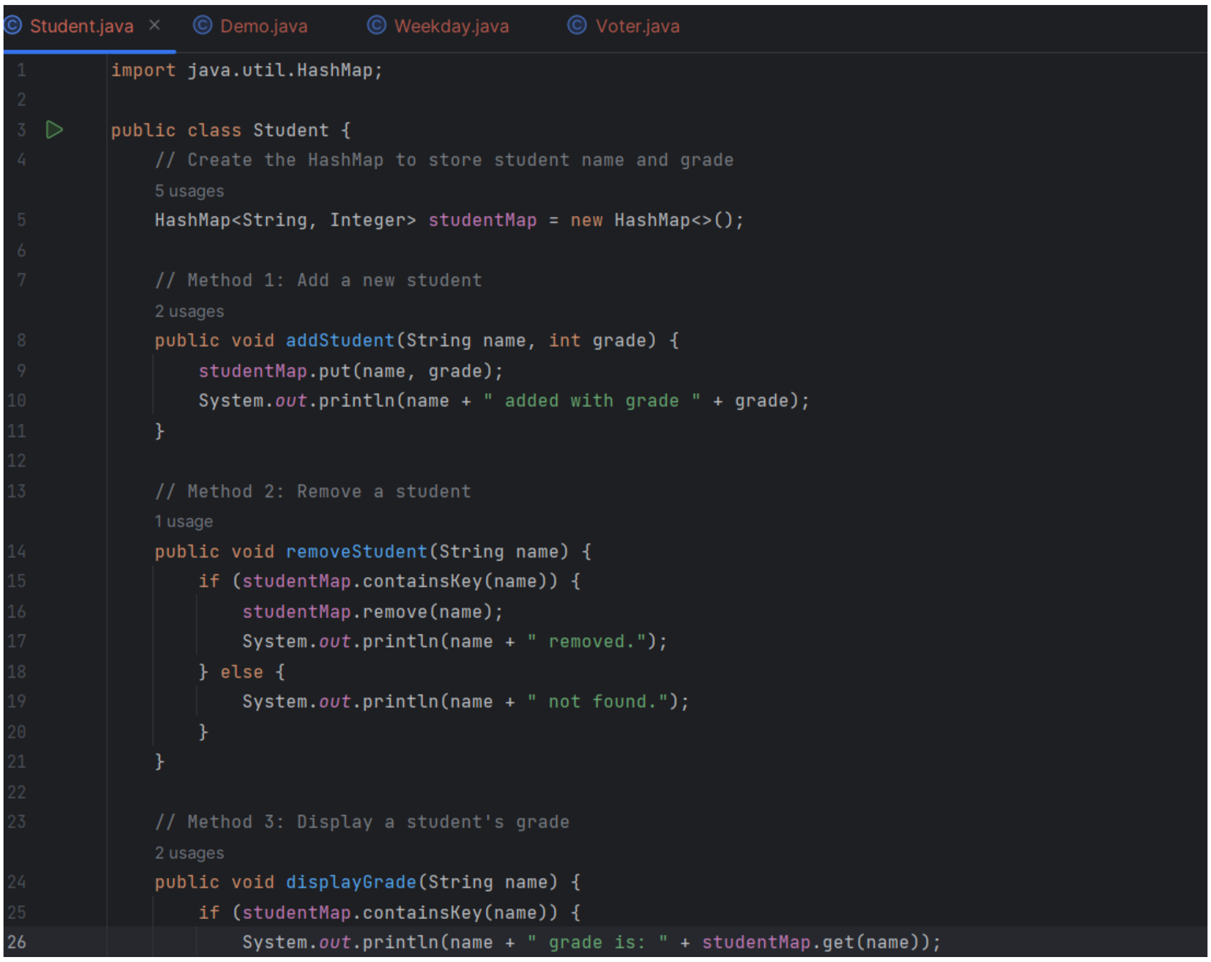
```java
import java.util.Scanner;

public class Weekday {
    public static void main(String[] args) {
        // Step 1: Create array with weekdays starting from Sunday
        String[] days = {"Sunday", "Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday"};

        // Step 2: Take index input from user
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter day index (0-6): ");
        int index = scanner.nextInt();

        // Step 3: Use try-catch to handle out-of-bound access
        try {
            System.out.println("Day is: " + days[index]);
        } catch (ArrayIndexOutOfBoundsException e) {
            System.out.println("Invalid index! Please enter a number between 0 and 6.");
        }
    }
}
```

Output:.



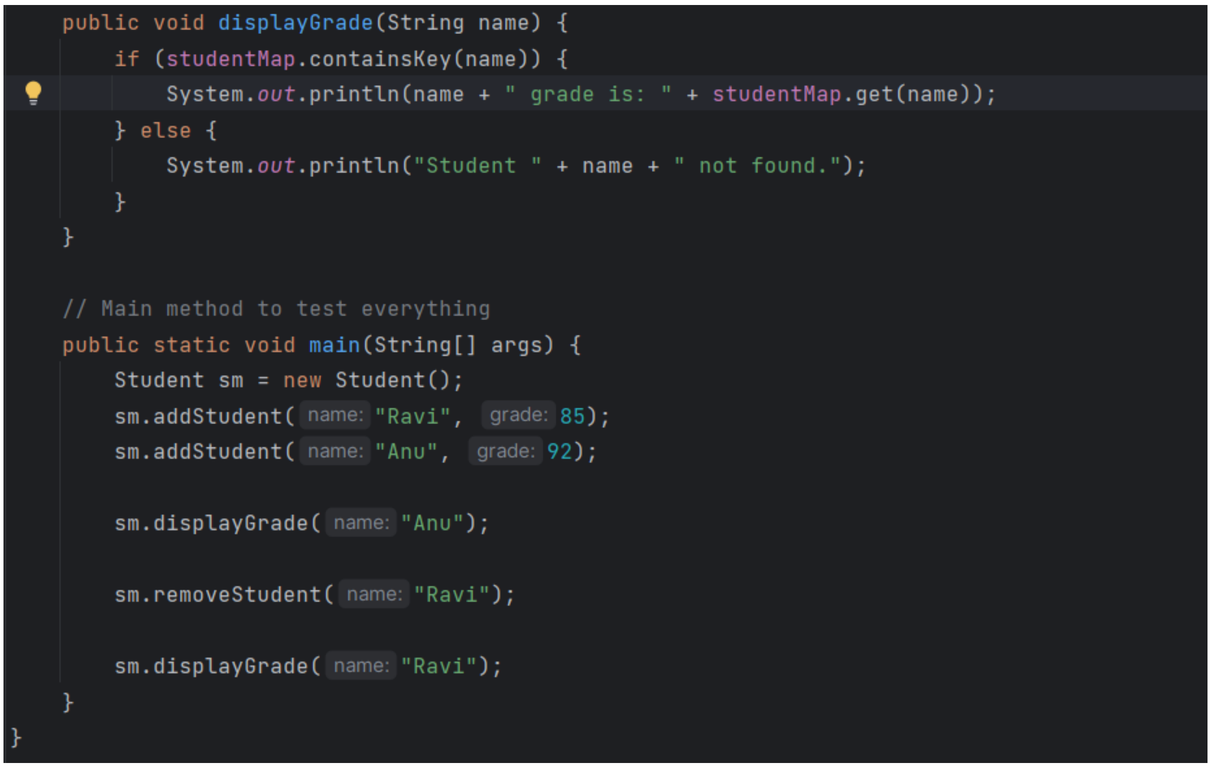
```
"C:\Program Files\Java\jdk-17.0.12\bin\java.exe" "-javaagent:C:\U
Enter day index (0-6): 1
Day is: Monday

Process finished with exit code 0
```

Q4. Create a HashMap where keys are student names (strings) and values are their corresponding grades (integers). Create methods to add a new student, remove a student, and Display up a student's grade by name.



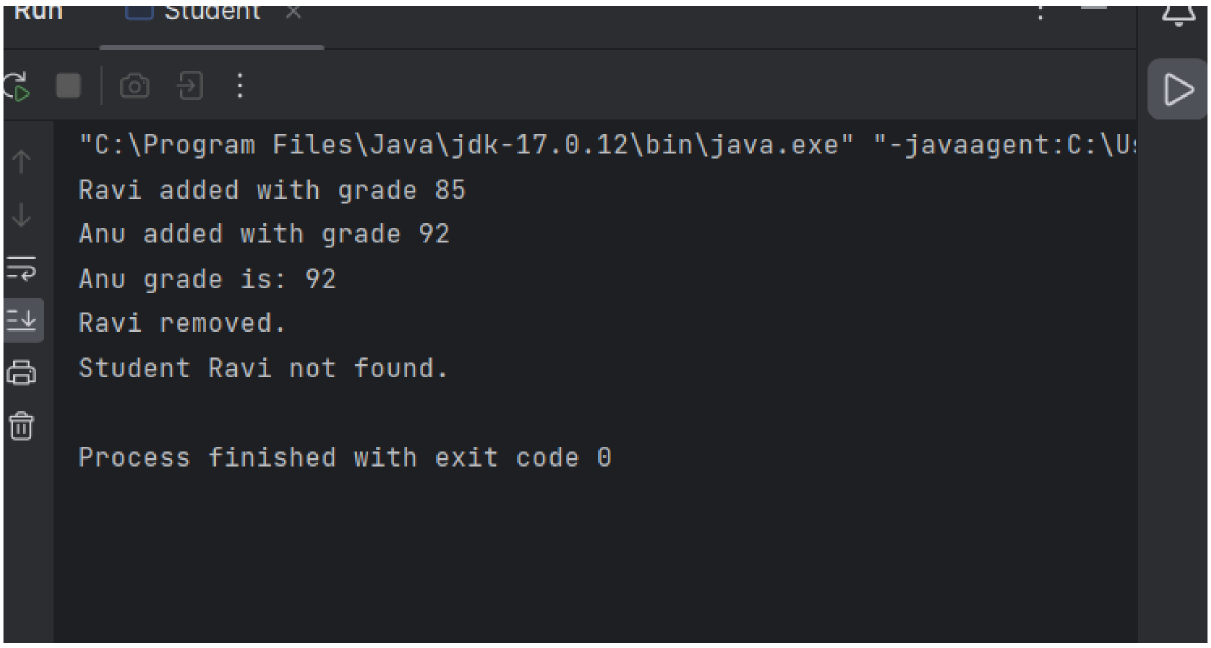
Student.java × Demo.java Weekday.java Voter.java

```java
import java.util.HashMap;

public class Student {
    // Create the HashMap to store student name and grade
    5 usages
    HashMap<String, Integer> studentMap = new HashMap<>();

    // Method 1: Add a new student
    2 usages
    public void addStudent(String name, int grade) {
        studentMap.put(name, grade);
        System.out.println(name + " added with grade " + grade);
    }

    // Method 2: Remove a student
    1 usage
    public void removeStudent(String name) {
        if (studentMap.containsKey(name)) {
            studentMap.remove(name);
            System.out.println(name + " removed.");
        } else {
            System.out.println(name + " not found.");
        }
    }

    // Method 3: Display a student's grade
    2 usages
    public void displayGrade(String name) {
        if (studentMap.containsKey(name)) {
            System.out.println(name + " grade is: " + studentMap.get(name));
```



```java
    public void displayGrade(String name) {
        if (studentMap.containsKey(name)) {
            System.out.println(name + " grade is: " + studentMap.get(name));
        } else {
            System.out.println("Student " + name + " not found.");
        }
    }

    // Main method to test everything
    public static void main(String[] args) {
        Student sm = new Student();
        sm.addStudent( name: "Ravi",  grade: 85);
        sm.addStudent( name: "Anu",  grade: 92);

        sm.displayGrade( name: "Anu");

        sm.removeStudent( name: "Ravi");

        sm.displayGrade( name: "Ravi");
    }
}
```

Output:



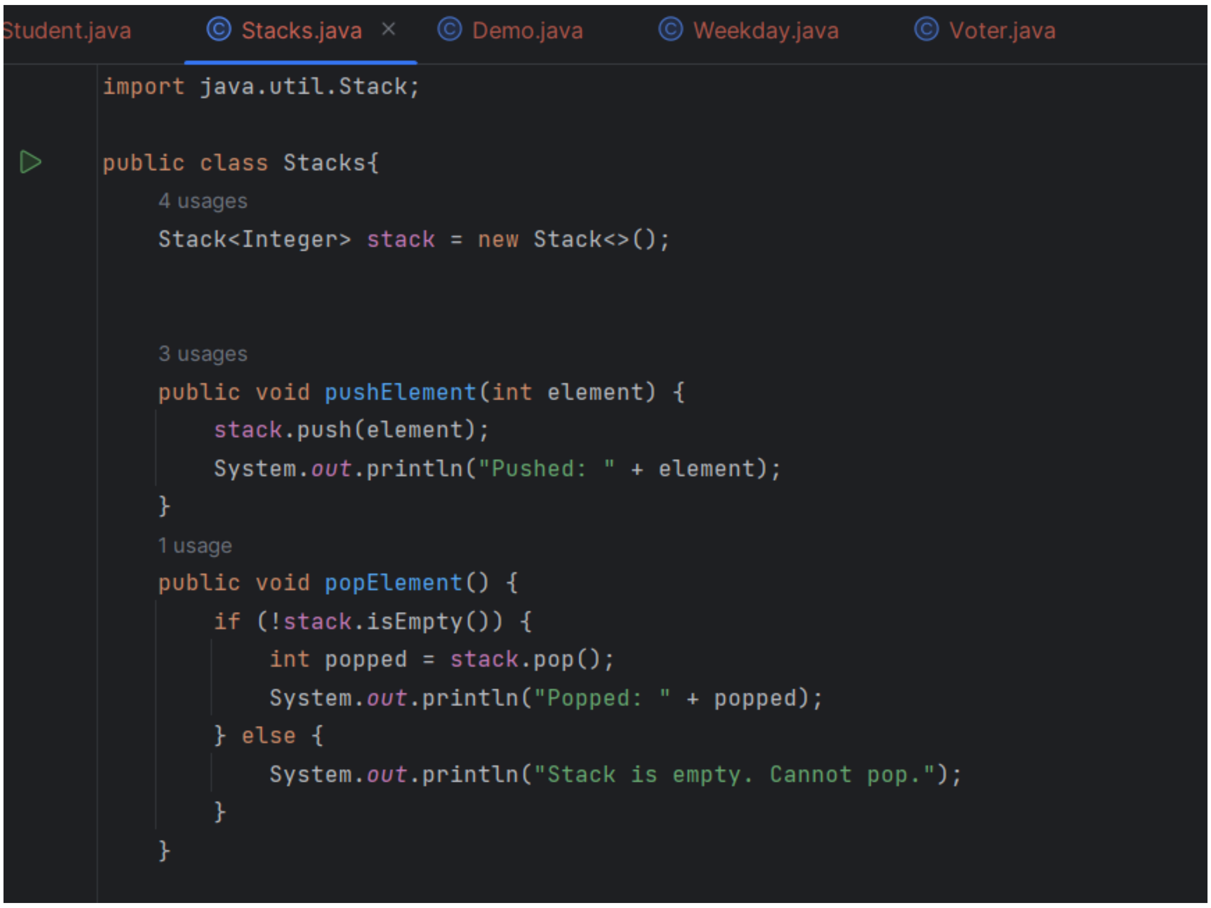
```
"C:\Program Files\Java\jdk-17.0.12\bin\java.exe" "-javaagent:C:\U
Ravi added with grade 85
Anu added with grade 92
Anu grade is: 92
Ravi removed.
Student Ravi not found.

Process finished with exit code 0
```

WeQ5. Use Collection Classes to store Integers.Create some methods for following functionalities.

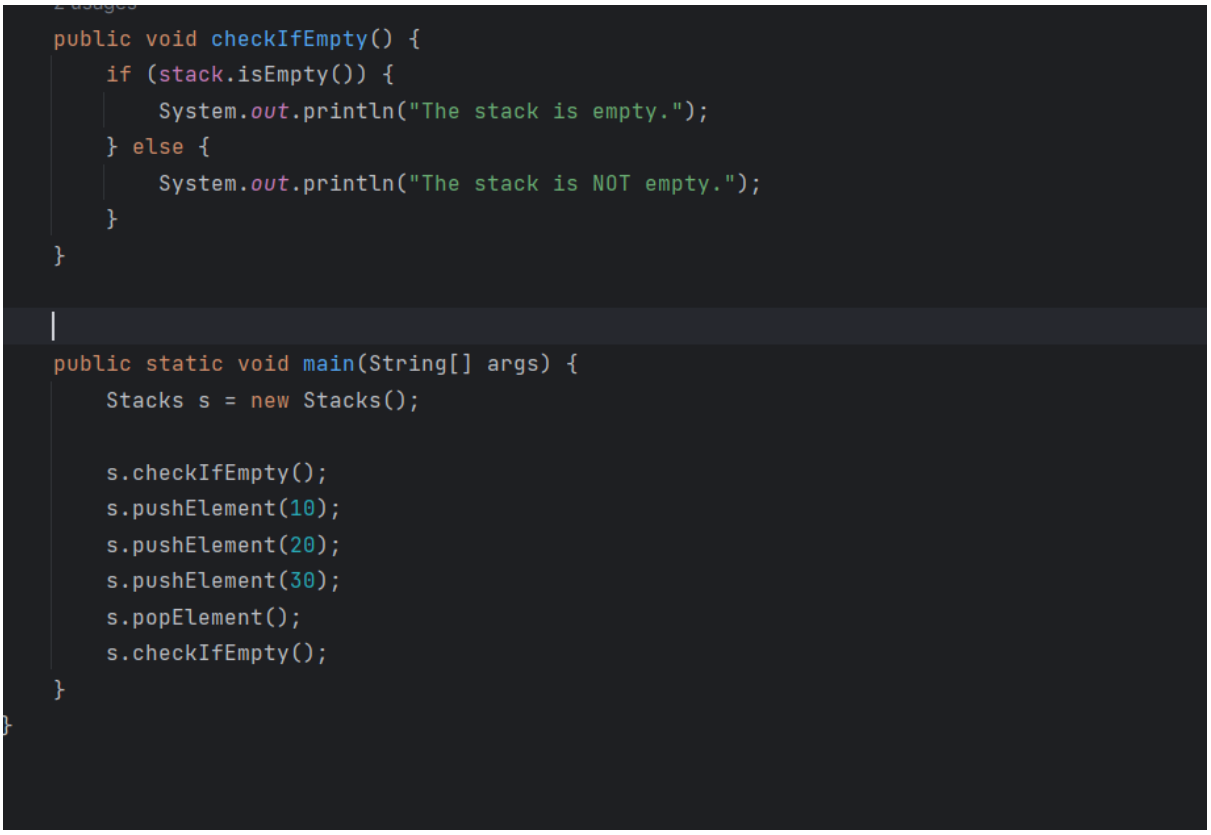
Include functions for pushing elements onto the stack.

popping elements from the stack.

Checking if the stack is empty

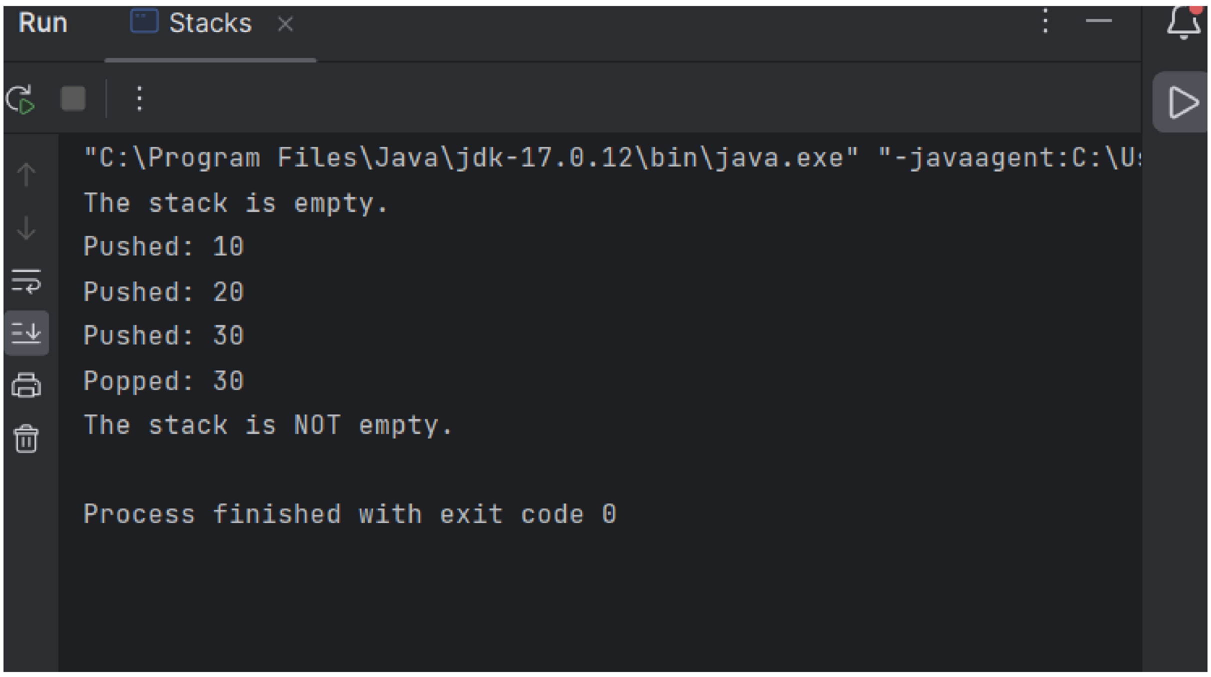


```
import java.util.Stack;

public class Stacks{
    4 usages
    Stack<Integer> stack = new Stack<>();


    3 usages
    public void pushElement(int element) {
        stack.push(element);
        System.out.println("Pushed: " + element);
    }
    1 usage
    public void popElement() {
        if (!stack.isEmpty()) {
            int popped = stack.pop();
            System.out.println("Popped: " + popped);
        } else {
            System.out.println("Stack is empty. Cannot pop.");
        }
    }
```



```
    public void checkIfEmpty() {
        if (stack.isEmpty()) {
            System.out.println("The stack is empty.");
        } else {
            System.out.println("The stack is NOT empty.");
        }
    }


    public static void main(String[] args) {
        Stacks s = new Stacks();

        s.checkIfEmpty();
        s.pushElement(10);
        s.pushElement(20);
        s.pushElement(30);
        s.popElement();
        s.checkIfEmpty();
    }
}
```

Output:



```
"C:\Program Files\Java\jdk-17.0.12\bin\java.exe" "-javaagent:C:\U
The stack is empty.
Pushed: 10
Pushed: 20
Pushed: 30
Popped: 30
The stack is NOT empty.

Process finished with exit code 0
```