Optionals

By Hatef Palizgar

Task 1

Create a method that takes a string parameter and returns an optional of Integer. The method should attempt to parse the string as an integer, and return an optional containing the parsed integer if successful, or an empty optional if the string cannot be parsed.

To test the method, you can use the following JUnit 5 code:

```
class TestTask1 {
   @Test
   void testParseInt() {
       Optional<Integer> numOpt1 = Optional.of(123);
       Optional<Integer> numOpt2 = Optional.empty();

       assertEquals(numOpt1, Task1.parseInt("123"));
       assertEquals(numOpt2, Task1.parseInt("abc"));
   }
}
```

Task 2

Create a class `Person` with the following fields: `name (String)`, `age (int)`, and `Optional<String> address`. Implement a constructor that takes the `name` and `age` as parameters, and initializes the `address` field to an empty optional. Implement getters and setters for all fields.

Task 3

Write a method that takes a list of `Person` objects and returns a list of the `names` of all the people with a non-empty address.

To test the method, you can use the following code:

```
class TestTask3 {
    @Test
    void testGetPeopleWithAddress() {
        // Given
        List<Person> people = new ArrayList<>();
        Person person1 = new Person("John", 30);
        person1.setAddress(Optional.of("123 Main St."));
        Person person2 = new Person("Jane", 25);
        person2.setAddress(Optional.empty());
        Person person3 = new Person("Bob", 40);
        person3.setAddress(Optional.empty());
        people.add(person1);
        people.add(person2);
        people.add(person3);

        // When
        List<String> names = Task3.getPeopleWithAddress(people);

        // Then
        Assertions.assertEquals(1, names.size());
        Assertions.assertTrue(names.contains("John"));
    }
}
```

Task 4

Write a method that takes an `Optional<Integer>` and returns a new optional containing the square of the integer value if it is present, or an empty optional if it is not present.

To test the method, you can use the following code:

```
class TestTask4 {
    @Test
    public void testSquareWithPresentValue() {
        // Given
        Optional<Integer> numOpt = Optional.of(5);

        // When
        Optional<Integer> resultOpt = Task4.square(numOpt);

        // Then
        Assertions.assertTrue(resultOpt.isPresent());
        Assertions.assertEquals(25, resultOpt.get());
    }

    @Test
    public void testSquareWithEmptyValue() {
        // Given
        Optional<Integer> numOpt = Optional.empty();

        // When
        Optional<Integer> resultOpt = Task4.square(numOpt);

        // Then
        Assertions.assertFalse(resultOpt.isPresent());
    }
}
```

Task 5

Write a method that takes a `List<Optional<String>>` and returns a new list containing all the non-empty string values, in the order they appear in the original list.

To test the method, you can use the following code:

```
class TestTask5 {
   @Test
   public void testGetNonEmptyStrings() {
        // Given
        List<Optional<String>> stringOpts = new ArrayList<>();
        stringOpts.add(Optional.of("hello"));
        stringOpts.add(Optional.of(""));
        stringOpts.add(Optional.empty());
        stringOpts.add(Optional.of("world"));

        // When
       List<String> nonEmptyStrings =
Task5.getNonEmptyStrings(stringOpts);

        // Then
        Assertions.assertEquals(3, nonEmptyStrings.size());
        Assertions.assertTrue(nonEmptyStrings.contains("hello"));
        Assertions.assertTrue(nonEmptyStrings.contains("world"));
        Assertions.assertTrue(nonEmptyStrings.contains(""));
   }
}
```

Task 6

Write a method that takes a `List<Integer>` and returns the maximum value in the list, or an empty optional if the list is empty.

To test the method, you can use the following code:

```
class TestTask6 {
    @Test
    public void testGetMaxWithNonEmptyList() {
        // Given
        List<Integer> numbers = Arrays.asList(1, 2, 5, 4, 3);

        // When
        Optional<Integer> result = Task6.getMax(numbers);

        // Then
        Assertions.assertTrue(result.isPresent());
        Assertions.assertEquals(5, result.get());
    }

    @Test
    public void testGetMaxWithEmptyList() {
        // Given
        List<Integer> numbers = new ArrayList<>();

        // When
        Optional<Integer> result = Task6.getMax(numbers);

        // Then
        Assertions.assertFalse(result.isPresent());
    }
}
```

Task 7

Write a method that takes an `Optional<String>` and returns the length of the string value if it is present, or -1 if it is not present.

To test the method, you can use the following code:

```
class TestTask7 {
    @Test
    public void testGetStringLengthWithPresentString() {
        // Given
        Optional<String> strOpt = Optional.of("hello world");

        // When
        int length = Task7.getStringLength(strOpt);

        // Then
        Assertions.assertEquals(11, length);
    }

    @Test
    public void testGetStringLengthWithEmptyString() {
        // Given
        Optional<String> strOpt = Optional.empty();

        // When
        int length = Task7.getStringLength(strOpt);

        // Then
        Assertions.assertEquals(-1, length);
    }
}
```

Task 8

Consider a `Person` class with a `name` and `age` field (The same `Person` class you created for **Task 2**). Write a method that takes a list of `Person` objects and returns the average `age` (`OptionalDouble` since `age` is of `double` type) of all people whose `names` start with a given prefix, or an empty optional if no such people are found.

To test the method, you can use the following code:

```
class TestTask8 {
    @Test
    public void testAverageAgeOfPeopleWithNamePrefixWithMatchingPrefix() {
        // Given
        List<Person> people = Arrays.asList(
                new Person("John", 30),
                new Person("Jane", 25),
                new Person("Joan", 40),
                new Person("Jim", 35)
        );
        String prefix = "Jo";

        // When
        OptionalDouble result = Task8.averageAgeOfPeopleWithNamePrefix(people,
prefix);

        // Then
        Assertions.assertTrue(result.isPresent());
        Assertions.assertEquals(35, result.getAsDouble());
    }

    @Test
    public void testAverageAgeOfPeopleWithNamePrefixWithNonMatchingPrefix() {
        // Given
        List<Person> people = Arrays.asList(
                new Person("John", 30),
                new Person("Jane", 25),
                new Person("Joan", 40),
                new Person("Jim", 35)
        );
        String prefix = "Dave";

        // When
        OptionalDouble result = Task8.averageAgeOfPeopleWithNamePrefix(people,
prefix);

        // Then
```

```
        Assertions.assertFalse(result.isPresent());
    }

    @Test
    public void testAverageAgeOfPeopleWithNamePrefixWithEmptyList() {
        // Given
        List<Person> people = new ArrayList<>();
        String prefix = "Jo";

        // When
        OptionalDouble result = Task8.averageAgeOfPeopleWithNamePrefix(people,
prefix);

        // Then
        Assertions.assertFalse(result.isPresent());
    }
}
```

Task 9

Write a method that takes a String and returns an `Optional<Integer>` representing the length of the String. If the input String is null, the method should return an empty Optional. Use `Optional` and `map()` method.

To test the method, you can use the following code:

```
class TestTask9 {
    @Test
    public void testGetStringLengthWithNonNullString() {
        // Given
        String str = "Hello, world!";

        // When
        Optional<Integer> result = Task9.getStringLength(str);

        // Then
        Assertions.assertTrue(result.isPresent());
        Assertions.assertEquals(13, result.get());
    }

    @Test
    public void testGetStringLengthWithNullString() {
        // Given
        String str = null;

        // When
        Optional<Integer> result = Task9.getStringLength(str);

        // Then
        Assertions.assertFalse(result.isPresent());
    }
}
```