گزارش آزمایشگاه مهندسی نرم‌افزار

پاییز ۱۴۰۱

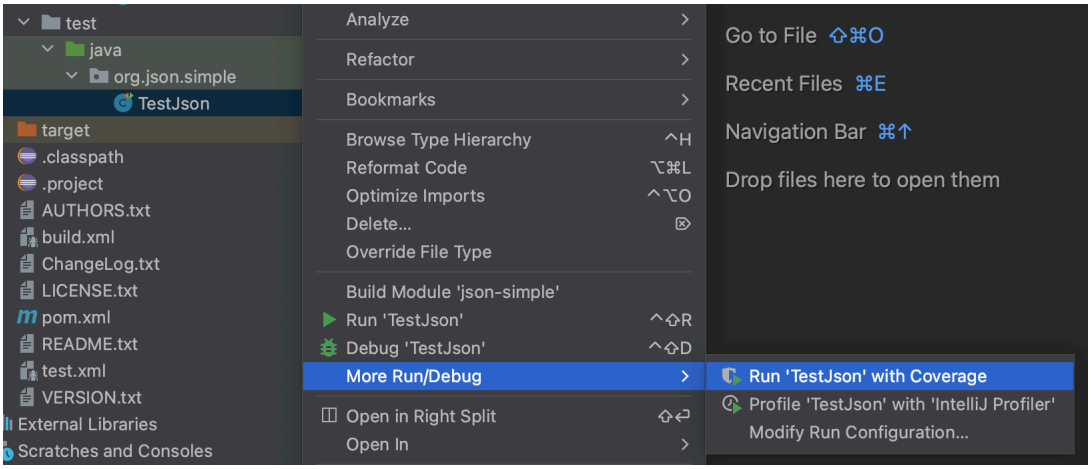
گزارش ۶: پوشش آزمون

دانشکده‌ی مهندسی کامپیوتر

مهرانه نجفی(۹۷۱۰۴۷۰۷)　　رستا روغنی(۹۷۱۰۵۹۶۳)

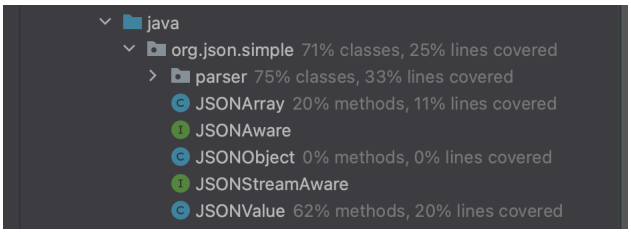
۱ پروژه:json-simple

همانطور که در دستورکار گفته‌شده‌است، پروژه را باز می‌کنیم و سپس کلاس TestJson را با گزینه‌ی Run with coverage اجرا می‌کنیم:

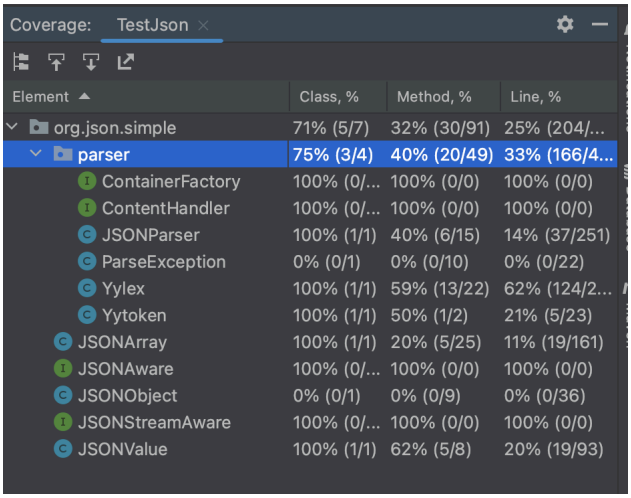


شکل ۱: Run json-simple with coverage

نتیجه‌ی اجرا به شکل زیر است که درصد خطها، توابع و کلاس‌هایی که از هرفایل توسط تست‌ها پوشش داده‌شده‌اند را نشان می‌دهد:



شکل ۲: Coverage result



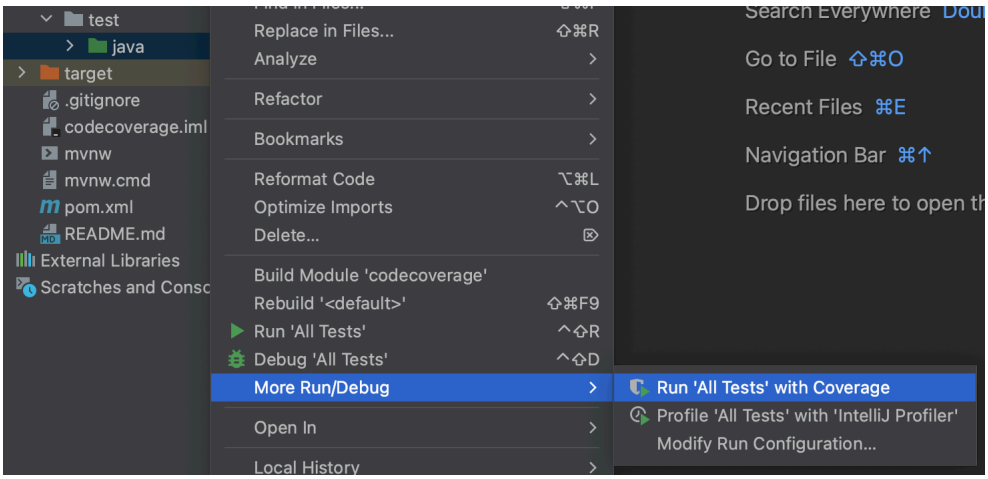
شکل ۳: Coverage result

همچنین با استفاده از گزینه‌ی Generate Coverage Report این گزارش را در فرمت html دریافت می‌کنیم. (درفولدر json-simple report فایل‌های مربوطه قرار گرفته‌اند)

۲ پروژه:CodeCoverageProject

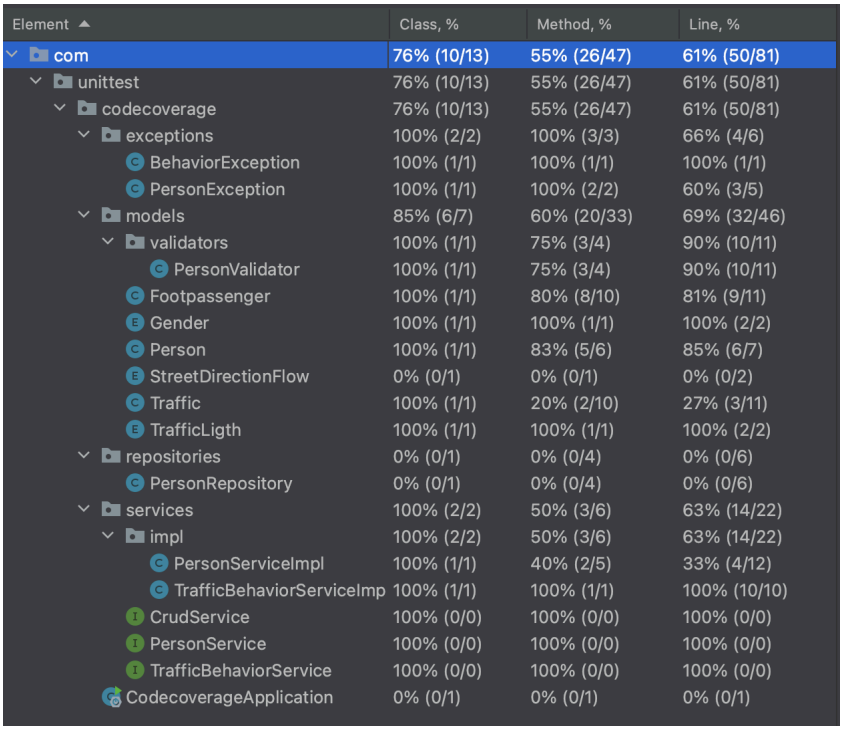
۱.۲ پروژه اولیه

همانند بخش قبل پروژه را باز کرده و از منوی Maven گزینه‌ی test را اجرا می‌کنیم تا dependency های لازم دانلود شوند. سپس گزینه‌ی Run All tests with coverage را انتخاب می‌کنیم.



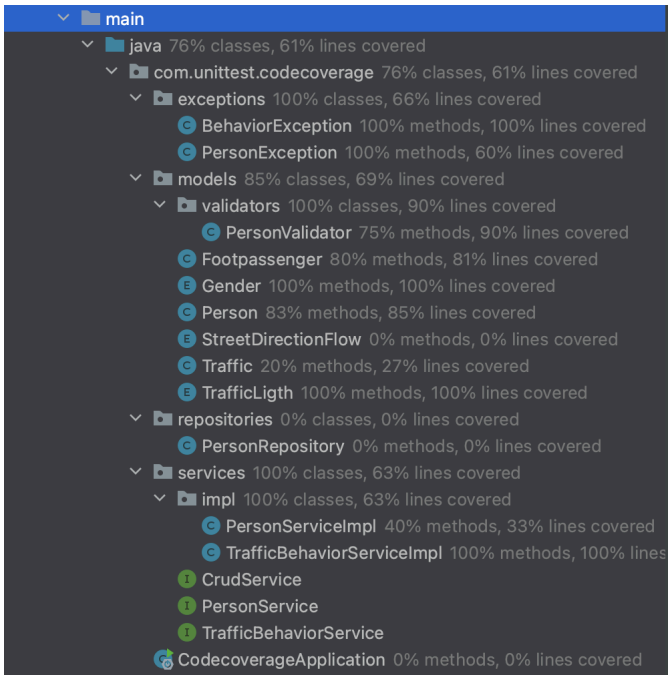
شکل ۴: Run All tests with coverage

نتیجه‌ی پوشش آزمون برای پروژه‌ی اولیه به شکل زیر است:



Element	Class, %	Method, %	Line, %
com	76% (10/13)	55% (26/47)	61% (50/81)
unittest	76% (10/13)	55% (26/47)	61% (50/81)
codecoverage	76% (10/13)	55% (26/47)	61% (50/81)
exceptions	100% (2/2)	100% (3/3)	66% (4/6)
BehaviorException	100% (1/1)	100% (1/1)	100% (1/1)
PersonException	100% (1/1)	100% (2/2)	60% (3/5)
models	85% (6/7)	60% (20/33)	69% (32/46)
validators	100% (1/1)	75% (3/4)	90% (10/11)
PersonValidator	100% (1/1)	75% (3/4)	90% (10/11)
Footpassenger	100% (1/1)	80% (8/10)	81% (9/11)
Gender	100% (1/1)	100% (1/1)	100% (2/2)
Person	100% (1/1)	83% (5/6)	85% (6/7)
StreetDirectionFlow	0% (0/1)	0% (0/1)	0% (0/2)
Traffic	100% (1/1)	20% (2/10)	27% (3/11)
TrafficLigth	100% (1/1)	100% (1/1)	100% (2/2)
repositories	0% (0/1)	0% (0/4)	0% (0/6)
PersonRepository	0% (0/1)	0% (0/4)	0% (0/6)
services	100% (2/2)	50% (3/6)	63% (14/22)
impl	100% (2/2)	50% (3/6)	63% (14/22)
PersonServiceImpl	100% (1/1)	40% (2/5)	33% (4/12)
TrafficBehaviorServiceImp	100% (1/1)	100% (1/1)	100% (10/10)
CrudService	100% (0/0)	100% (0/0)	100% (0/0)
PersonService	100% (0/0)	100% (0/0)	100% (0/0)
TrafficBehaviorService	100% (0/0)	100% (0/0)	100% (0/0)
CodecoverageApplication	0% (0/1)	0% (0/1)	0% (0/1)

شکل ۵: Coverage result



شکل ۶: Coverage result

گزارش نتایج را در فرمت html نیز دریافت می‌کنیم. (درفولدر CodeCoverageProject initial report فایل‌های مربوطه قرار گرفته‌اند)

۲.۲ پروژه نهایی

در این بخش به نوشتن تست برای بهبود درصد پوشش همه‌ی کلاس‌ها می‌پردازیم.

۱.۲.۲ PersonException

خطوط قرمز تست نشده‌اند:

```
public class PersonException extends RuntimeException {

	private static final long serialVersionUID = 1L;

	private List<String> errors = new ArrayList<>();

	public PersonException(String msg) {
		super(msg);
		this.errors.add(msg);
	}

	public PersonException(List<String> errors) {
		super(errors != null && !errors.isEmpty()? String.join(";", errors) : null);
		this.errors = errors;
	}

}
```

شکل ۷: PersonException Class

تست مربوطه را در بخش PersonServiceImpl توضیح می‌دهیم. در این تست توابع get و delete موجود در کلاس PersonServiceImpl اجرا می‌شوند که با استفاده از constructor بالا و ورودی دادن پیام دلخواه به شکل یک رشته، exception جدید تولید می‌کنند. درنتیجه این تابع هم مورد آزمون قرار می‌گیرد. درصدها پس از اجرای آزمون به شکل زیر تغییر می‌کنند:

(آ) قبل

(ب) بعد

شکل ۸: Result

۲.۲.۲ PersonValidator

خطوط قرمز تست نشده‌اند:

```
public class PersonValidator {

	public boolean requiredName(Person person) {
		return person != null && person.getName() != null && person.getName().trim().length() > 0;
	}

	public boolean requiredName(String name) {
		return name != null && name.trim().length() > 0;
	}

	public boolean requiredGender(Person person) {
		return person != null && person.getGender() != null;
	}

	public void validate(Person person) {
		List<String> errors = new ArrayList<>();
		if(!requiredName(person)) {
			errors.add("Name is required");
		}

		if(!requiredGender(person)) {
			errors.add("Gender is required");
		}

		if (!errors.isEmpty()) {
			throw new PersonException(errors);
		}
	}
```

شکل ۹: PersonValidator Class

تست مربوطه در بخش PersonServiceImpl قرار دارد. همانطور که گفته شد در این تست توابع get و delete استفاده می‌شوند که در خودشان requiredName را برای سنجش معتبر بودن اسم فرد، صدا می‌کنند. بنابراین این تابع هم در تست مذکور بررسی می‌شود.درصدها پس از اجرای آزمون به شکل زیر تغییر می‌کنند:

(آ) قبل (ب) بعد

شکل ۱۰: Result

۳.۲.۲ Footpassenger

خطوط قرمز تست نشده‌اند:

```
public class Footpassenger {

	private boolean crossedTheCrosswalk;
	private TrafficLigth crossedTrafficLigth;
	private boolean lookedToTheRight;
	private boolean lookedToTheLeft;
	private boolean crossedTheRoad;

	public boolean crossedTheCrosswalk() {
		return crossedTheCrosswalk;
	}
	public void setCrossedTheCrosswalk(boolean crossedTheCrosswalk) {
		this.crossedTheCrosswalk = crossedTheCrosswalk;
	}
	public TrafficLigth getCrossedTrafficLigth() {
		return crossedTrafficLigth;
	}
	public void setCrossedTrafficLigth(TrafficLigth crossedSignaling) {
		this.crossedTrafficLigth = crossedSignaling;
	}
	public boolean lookedToTheRight() {
		return lookedToTheRight;
	}
	public void setLookedToTheRight(boolean lookedToTheRight) {
		this.lookedToTheRight = lookedToTheRight;
	}
	public boolean lookedToTheLeft() {
		return lookedToTheLeft;
	}
	public void setLookedToTheLeft(boolean lookedToTheLeft) {
		this.lookedToTheLeft = lookedToTheLeft;
	}
	public boolean crossedTheRoad() {
		return crossedTheRoad;
	}
	public void setCrossedTheRoad(boolean crossedTheStreet) {
		this.crossedTheRoad = crossedTheStreet;
	}


}
```

شکل ۱۱: Footpassenger Class

پس رد شدن از خط عابرپیاده بررسی نشده‌است. تستی می‌نویسیم که عابر پیاده به هنگام سبز بودن چراغ، با نگاه کردن به راست و چپ از خط عابرپیاده رد می‌شود و نباید هیچ exception ای دریافت کند. علاوه‌بر این تست می‌کنیم که پس از اینکه عبور از خط پیاده را true کردیم، مقدار صحیحی در فیلد مربوطه ذخیره شده‌باشد.

```java
@Test
@DisplayName("Shouldn't do anything when footpassenger crosses the crossroad when the traffic light is green and pays attention")
public void testFootpassengerCrossTheCrossroad_shouldDoNothingWhenFootpassengerCrossesTheCrossroadDuringGreenLightAndLookSides() {

    Traffic currentTrafic = new Traffic();
    currentTrafic.setIntenseCarTraffic(true);

    Footpassenger currentFootpassengerBehavior = new Footpassenger();
    currentFootpassengerBehavior.setCrossedTheCrosswalk(true);
    currentFootpassengerBehavior.setCrossedTrafficLigth(TrafficLigth.GREEN);
    currentFootpassengerBehavior.setLookedToTheLeft(true);
    currentFootpassengerBehavior.setLookedToTheRight(true);

    assertTrue(currentFootpassengerBehavior.crossedTheCrosswalk());

    assertAll(() -> trafficBehaviorService.footpassengerCrossTheStreet(currentTrafic, currentFootpassengerBehavior));
}
```

شکل ۱۲: Test

درصدها پس از اجرای آزمون به شکل زیر تغییر می‌کنند:

(آ) قبل (ب) بعد

شکل ۱۳: Result

۴.۲.۲ Person

خطوط قرمز تست نشده‌اند:

```java
public class Person implements Serializable {

	private static final long serialVersionUID = 1L;

	private String name;
	private int age;
	private Gender gender;

	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 Gender getGender() {
		return gender;
	}
	public void setGender(Gender gender) {
		this.gender = gender;
	}

}
```

شکل ۱۴: Person Class

یکی از تست‌ها را به‌گونه‌ای تغییر می‌دهیم که بعد از تعیین کردن سن، people نتیجه‌ی getAge را چک کند تا با همان سنی که ورودی داده‌است برابر باشد. علاوه بر این در تست بخش PersonRepository هم از این تابع استفاده می‌کنیم.

```java
public void testInsert_shouldInsertPersonWithSuccessWhenAllPersonsInfoIsFilled() {
    Person person = new Person();
    person.setName("Name");
    person.setAge(21);
    person.setGender(Gender.M);

    assertEquals(person.getAge(), actual: 21);

    when(repository.insert(any(Person.class))).thenReturn(person);

    service.insert(person);
}
```

شکل ۱۵: Test

درصدها پس از اجرای آزمون به شکل زیر تغییر می‌کنند:

(ب) بعد

(آ) قبل

شکل ۱۶: Result

StreetDirectionFlow ۵.۲.۲

خطوط قرمز تست نشده‌اند:

```java
package com.unittest.codecoverage.models;

public enum StreetDirectionFlow {

        ONE_WAY, TWO_WAY;

}
```

شکل ۱۷: StreetDirectionFlow Class

پس جهت‌دار بودن خیابان بررسی نشده‌است. برای اینکه تست‌ها معناداری باشند، دو تست مشترک برای این بخش و بخش بعدی می‌نویسیم که آن را در ادامه توضیح می‌دهیم. درصدها پس از اجرای آزمون به شکل زیر تغییر می‌کنند:

(ب) بعد

(آ) قبل

شکل ۱۸: Result

Traffic ۶.۲.۲

خطوط قرمز تست نشده‌اند:

```java
public class Traffic {

	private TrafficLigth currentTrafficLight;
	private boolean intenseCarTraffic;
	private short maxSpeedAllowed;
	private short minSpeedAllowed;
	private StreetDirectionFlow streetDirectionFlow;

	public TrafficLigth getCurrentTrafficLight() {
		return currentTrafficLight;
	}
	public void setCurrentTrafficLight(TrafficLigth currentSignaling) {
		this.currentTrafficLight = currentSignaling;
	}
	public boolean intenseCarTraffic() {
		return intenseCarTraffic;
	}
	public void setIntenseCarTraffic(boolean intenseCarTraffic) {
		this.intenseCarTraffic = intenseCarTraffic;
	}
	public short getMaxSpeedAllowed() {
		return maxSpeedAllowed;
	}
	public void setMaxSpeedAllowed(short maxSpeedAllowed) {
		this.maxSpeedAllowed = maxSpeedAllowed;
	}
	public short getMinSpeedAllowed() {
		return minSpeedAllowed;
	}
	public void setMinSpeedAllowed(short minSpeedAllowed) {
		this.minSpeedAllowed = minSpeedAllowed;
	}
	public StreetDirectionFlow getStreetDirectionFlow() {
		return streetDirectionFlow;
	}
	public void setStreetDirectionFlow(StreetDirectionFlow streetDirectionFlow) {
		this.streetDirectionFlow = streetDirectionFlow;
	}

}
```

شکل ۱۹: Traffic Class

ابتدا یکی از تست‌ها را تغییر می‌دهیم و در آن خیابان را دوطرفه در نظر می‌گیریم و چون چراغ عابر پیاده سبز است، چراغ ماشین را قرمز قرار می‌دهیم. در این شرایط اگر عابر بدون نگاه کردن به چپ و راست از خیابان عبور کند، باید exception ای با این مضمون دریافت کند که بیشتر مراقب باشد. همچنین بعد از تعیین کردن هریک از مقادیر، چک می‌کنیم که درون فیلدها مقدار درستی ذخیره‌شده‌باشد.

```java
@Test
@DisplayName("Should throw exception when footpassenger crosses the road during heavy traffic without attention")
public void testFootpassengerCrossTheStreet_shouldThrowBehaviorExceptionWhenFootpassengerCrossesTheRoadDuringHeavyTrafficWithoutLo

    Traffic currentTrafic = new Traffic();
    currentTrafic.setIntenseCarTraffic(true);
    currentTrafic.setStreetDirectionFlow(StreetDirectionFlow.TWO_WAY);
    currentTrafic.setCurrentTrafficLight(TrafficLigth.RED);

    Footpassenger currentFootpassengerBehavior = new Footpassenger();
    currentFootpassengerBehavior.setCrossedTheRoad(true);
    currentFootpassengerBehavior.setCrossedTrafficLigth(TrafficLigth.GREEN);
    currentFootpassengerBehavior.setLookedToTheLeft(false);
    currentFootpassengerBehavior.setLookedToTheRight(false);

    assertEquals(currentTrafic.getCurrentTrafficLight(), TrafficLigth.RED);
    assertEquals(currentTrafic.getStreetDirectionFlow(), StreetDirectionFlow.TWO_WAY);

    Assertions.assertThatThrownBy(() -> trafficBehaviorService.footpassengerCrossTheStreet(currentTrafic, currentFootpassengerBeha
        .isInstanceOf(BehaviorException.class) capture of ?
        .hasMessage("You should be more careful");

}
```

شکل ۲۰: Test

همچنین تستی می‌نویسیم که در آن خیابانی یک طرفه با ترافیک سبک داریم (سرعت ماشین‌ها را حداقل ۲۰ و حداکثر ۴۰ قرار می‌دهیم) و چراغ عابر پیاده زرد است. در این شرایط عابر باید بتواند بدون دریافت exception از خیابان عبور کند.

```java
@Test
@DisplayName("Shouldn't do anything when footpassenger crosses the one-way street when the traffic light is yellow and traffic is
public void testFootpassengerCrossTheRoad_shouldDoNothingWhenFootpassengerCrossesTheRoadDuringYellowLightAndNotIntenseTraffic() {

    Traffic currentTrafic = new Traffic();
    currentTrafic.setIntenseCarTraffic(false);
    currentTrafic.setStreetDirectionFlow(StreetDirectionFlow.ONE_WAY);
    currentTrafic.setMinSpeedAllowed((short) 20);
    currentTrafic.setMaxSpeedAllowed((short) 40);

    Footpassenger currentFootpassengerBehavior = new Footpassenger();
    currentFootpassengerBehavior.setCrossedTheRoad(true);
    currentFootpassengerBehavior.setCrossedTrafficLigth(TrafficLigth.YELLOW);

    assertEquals(currentTrafic.getMinSpeedAllowed(),  actual: 20);
    assertEquals(currentTrafic.getMaxSpeedAllowed(),  actual: 40);

    assertAll(() -> trafficBehaviorService.footpassengerCrossTheStreet(currentTrafic, currentFootpassengerBehavior));
}
```

شکل ۲۱: Test

درصدها پس از اجرای آزمون به شکل زیر تغییر می‌کنند:

(آ) قبل (ب) بعد

شکل ۲۲: Result

۷.۲.۲ PersonRepository

خطوط قرمز تست نشده‌اند:

```java
public class PersonRepository {

        public Person insert(Person person) {
                Objects.requireNonNull(person, "person can't be null");
                return person;
        }

        public void update(Person person) {
                Objects.requireNonNull(person, "person can't be null");
        }

        public void delete(String name) {
                Objects.requireNonNull(name, "name can't be null");
        }

        public Person get(String name) {
                Objects.requireNonNull(name, "name can't be null");
                return null;
        }
```

شکل ۲۳: PersonRepository Class

برای اینکه مستقیما خود ریپازیتوری را تست کنیم، یک شی ریپازیتوری جدید می‌سازیم و تلاش می‌کنیم برای یک شی people که اطلاعاتش کامل است، عملیات‌های insert و get و update و delete را روی ریپازیتوری‌اش انجام دهیم که در این حالت عملیات‌ها باید بدون خطا اجرا شوند. حالت دیگر این است که به این توابع ورودی‌های null بدهیم که

در این صورت همگی آن‌ها باید exception بدهند چون تولید یک ریپازیتوری برای شی people ای که وجود ندارد، امکان‌پذیر نیست.

```java
public void testRepository_shouldInsertAndGetAndUpdateAndDeletePersonWithSuccessWhenAllPersonsInfoIsFilled(
    Person person = new Person();
    person.setName("Name");
    person.setAge(21);
    person.setGender(Gender.M);

    PersonRepository newRepo = new PersonRepository();

    //Insert
    newRepo.insert(person);
    assertThatThrownBy(() -> newRepo.insert( person: null))
            .hasMessage("person can't be null");

    //Update
    newRepo.update(person);
    assertThatThrownBy(() -> newRepo.update( person: null))
            .hasMessage("person can't be null");

    //Get
    newRepo.get(person.getName());
    assertThatThrownBy(() -> newRepo.get(null))
            .hasMessage("name can't be null");

    //Delete
    newRepo.delete(person.getName());
    assertThatThrownBy(() -> newRepo.delete( name: null))
            .hasMessage("name can't be null");
```

شکل ۲۴: Test

درصدها پس از اجرای آزمون به شکل زیر تغییر می‌کنند:

(آ) قبل

(ب) بعد

شکل ۲۵: Result

۸.۲.۲ PersonServiceImpl

خطوط قرمز تست نشده‌اند:

```java
@Service
public class PersonServiceImpl implements PersonService {

	private PersonValidator validator;
	@Autowired
	private PersonRepository repository;

	@Autowired
	public PersonServiceImpl() {
		this.validator = new PersonValidator();
	}

	@Override
	public Person insert(Person person) {
		validator.validate(person);
		return repository.insert(person);
	}

	@Override
	public void update(Person person) {
		validator.validate(person);
		repository.update(person);
	}

	@Override
	public Person get(String name) {
		if(validator.requiredName(name)) {
			throw new PersonException("Name is required");
		}
		return repository.get(name);
	}

	@Override
	public void delete(String name) {
		if(validator.requiredName(name)) {
			throw new PersonException("Name is required");
		}
		repository.delete(name);
	}

}
```

شکل ۲۶: PersonServiceImpl Class

باید تستی بنویسیم که در آن متدهای بالا را به کار ببریم. ابتدا یک شی جدید از people با اطلاعات کامل می‌سازیم. سپس دستور آپدیت را اجرا می‌کنیم که طبق تعاریف چون به آن ورودی کامل داده‌ایم، بدون خطا اجرا می‌شود اما اگر همین متد را با ورودی null اجرا کنیم، بخاطر تعریف نشدن اسم و جنسیت دو exception می‌خوریم. همچنین اگر به توابع get و delete اسم صحیح ورودی بدهیم، خروجی exception می‌دهند (احتمالا یکی از ایرادات کد است که با تست مشخص شده‌است) و درمقابل اگر به آن‌ها null ورودی بدهیم خطایی نمی‌گیریم که هردو حالت را در این تست بررسی می‌کنیم.

```java
public void testPersonServiceImpl_shouldDeleteAndUpdateAndGetPersonWithSuccessWhenAllPersonsInfoIsFilled()

    Person person = new Person();
    person.setName("Name");
    person.setAge(21);
    person.setGender(Gender.M);

    //Update
    service.update(person);
    assertThatThrownBy(() -> service.update( object: null)) AbstractThrowableAssert<capture of ?, capture of ? extends Throwa
            .isInstanceOf(PersonException.class) capture of ?
            .hasMessage("Name is required;Gender is required");

    //Get
    assertThatThrownBy(() -> service.get("Name")) AbstractThrowableAssert<capture of ?, capture of ? extends Throwable>
            .isInstanceOf(PersonException.class) capture of ?
            .hasMessage("Name is required");
    service.get(null);

    //Delete
    assertThatThrownBy(() -> service.delete( id: "Name")) AbstractThrowableAssert<capture of ?, capture of ? extends Throwabl
            .isInstanceOf(PersonException.class) capture of ?
            .hasMessage("Name is required");
    service.delete( id: null);
```

شکل ۲۷: Test

درصدها پس از اجرای آزمون به شکل زیر تغییر می‌کنند:

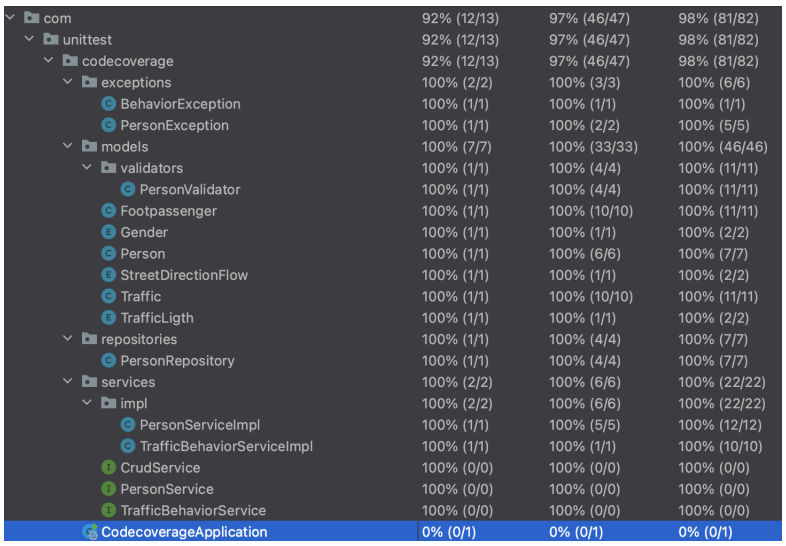
(آ) قبل

(ب) بعد

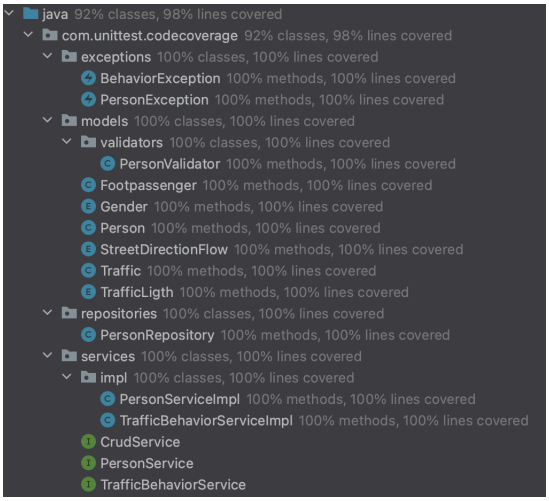
شکل ۲۸: Result

۹.۲.۲ نتایج نهایی

بعد از اینکه تمامی تست‌ها را به پروژه اضافه کردیم، یک دور دیگر Run All tests with coverage را اجرا می‌کنیم که نتیجه‌اش به شکل زیر است:



شکل ۲۹: Coverage result



شکل ۳۰: Coverage result

تمامی کلاس‌ها صد در صد تحت تست قرار گرفته‌اند (کلاس CodecoverageApplication هم که در خود تست‌ها را اجرا می‌کند). گزارش نتایج نهایی را در فرمت html نیز دریافت می‌کنیم. (درفولدر CodeCoverageProject final report فایل‌های مربوطه قرار گرفته‌اند)

در ریپازیتوری زیر هم تمامی کدها و گزارش کار آزمایش قرار داده‌شده‌است.

https://github.com/Miraneh/SoftwareLab