INGENIERIA DEL SOFTWARE II

Laboratorio

Testing y Mantenimiento

Equipo de trabajo: G2.04

AUTORES

DNI	Apellidos, Nombre	Correo UCLM
53406191-F	Escalona Arroyo, Marcos*	marcos.escalona@alu.uclm.es
53579547-N	Lopez Nieto, Carlos	carlos.lopez16@alu.uclm.es
71359733-W	Gutierrez Duran, Josué	josue.gutierrez1@alu.uclm.es
71559406-N	Gonzalez Sanjosé, Jesus Manuel	jesusmanuel.gonzalez1@alu.uclm.es

* Coordinador del grupo

CIUDAD REAL, DICIEMBRE DE 2016

CONTENIDO

TESTING Y MANTENIMIENTO

Para llevar a cabo las pruebas de Testing, utilizamos el código facilitado por el profesor Macario Polo, creando el proyecto "MultasSinWebNiMaven" según sus indicaciones.

A) CASOS DE PRUEBA Y ERRORES QUE ÉSTOS HAN ENCONTRADO

Creamos la clase "***TestManager.java***" en la que incluiremos todos los casos de prueba que creamos para efectuar el "testing" del código. Comprobaremos que el importe de las multas y los puntos a detraer de acuerdo a la velocidad del vehículo sean correctos para todos los casos. Para ello, optamos por realizar casos de prueba unitarios para cada comprobación, es decir, no estarán metidos en ningún bucle. Serán los siguientes casos (valores límite e intermedios por cada bloque de velocidades):

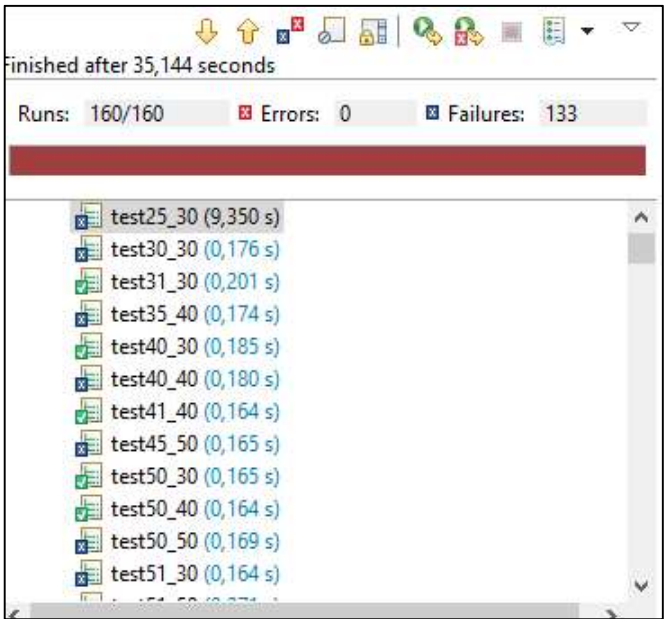
La velocidad máxima es	Comprobamos las siguientes velocidades															
30	25	30	31	40	50	51	55	60	61	65	70	71	75	80	81	85
40	35	40	41	50	60	61	65	70	71	75	80	81	85	90	91	95
50	45	50	51	60	70	71	75	80	81	85	90	91	95	100	101	105
60	55	60	61	70	90	91	100	110	111	115	120	121	125	130	131	135
70	65	70	71	85	100	101	110	120	121	125	130	131	135	140	141	145
80	75	80	81	95	110	111	120	130	131	135	140	141	145	150	151	155
90	85	90	91	105	120	121	130	140	141	145	150	151	155	160	161	165
100	95	100	101	115	130	131	140	150	151	155	160	161	165	170	171	175
110	105	110	111	125	140	141	150	160	161	165	170	171	175	180	181	185
120	115	120	121	135	150	151	160	170	171	175	180	181	185	190	191	195

Y realizamos todos ellos con la misma estructura de método que se muestra a continuación, variando solo, en cada caso, los datos que se indican:

```
@Test
public void test55_30() throws Exception {
    Manager m=Manager.get();

    DriverDao dao=new DriverDao();
    Driver driver=dao.findByDni("5000000");
    int pointsAntes=driver.getPoints();

    int idInquiry=m.openInquiry("0000",55,"Murcia",30);
    Sanction sanction=m.identifyDriver(idInquiry,"5000000");

    m.pay(sanction.getId());
    driver=dao.findByDni("5000000");
    assertTrue(driver.getPoints()==pointsAntes-2);//
    assertTrue(sanction.getAmount()==300);
}
```

55 → aquí la velocidad a la que circula el vehículo en la prueba.
30 → velocidad máxima del radar.
2 → puntos a detraer. En caso de no haber detracción no figura cantidad.
300 → cuantía de la sanción.

Ejecutando nuestros casos de prueba obtenemos los siguientes resultados:



Y, según “Eclemma”, nuestra cobertura es la siguiente:

Element	Coverage	Covered Instructio...	Missed Instructions	Total Instructions
src	79,8 %	7.779	1.969	9.748
edu.uclm.esi.iso2.multas.test	80,2 %	6.494	1.599	8.093
edu.uclm.esi.iso2.multas.domain	85,8 %	1.133	188	1.321
SanctionHolder.java	5,7 %	3	50	53
Inquiry.java	96,3 %	1.005	39	1.044
Manager.java	70,8 %	63	26	89
Sanction.java	58,6 %	34	24	58
Owner.java	38,2 %	13	21	34
Vehicle.java	26,1 %	6	17	23
Driver.java	45,0 %	9	11	20
edu.uclm.esi.iso2.multas.dao	45,5 %	152	182	334
GeneralDao.java	54,5 %	60	50	110
OwnerDao.java	42,3 %	33	45	78
HibernateFactory.java	42,6 %	29	39	68
VehicleDao.java	0,0 %	0	39	39
DriverDao.java	76,9 %	30	9	39

Antes de hacer el testing hemos realizado limpieza, eliminando el “código muerto” que hemos localizado.

Analizando los resultados del testing efectuado, vemos que los errores se localizan en los casos de prueba marcados en rojo:

La velocidad máxima es	Comprobamos las siguientes velocidades															
30	25	30	31	40	50	51	55	60	61	65	70	71	75	80	81	85
40	35	40	41	50	60	61	65	70	71	75	80	81	85	90	91	95
50	45	50	51	60	70	71	75	80	81	85	90	91	95	100	101	105
60	55	60	61	70	90	91	100	110	111	115	120	121	125	130	131	135
70	65	70	71	85	100	101	110	120	121	125	130	131	135	140	141	145
80	75	80	81	95	110	111	120	130	131	135	140	141	145	150	151	155
90	85	90	91	105	120	121	130	140	141	145	150	151	155	160	161	165
100	95	100	101	115	130	131	140	150	151	155	160	161	165	170	171	175
110	105	110	111	125	140	141	150	160	161	165	170	171	175	180	181	185
120	115	120	121	135	150	151	160	170	171	175	180	181	185	190	191	195

B) CAMBIOS REALIZADOS PARA CORREGIR ESOS ERRORES

Realizamos los siguientes cambios en el código:

- En la clase “Inquiry”, método “createSanctionFor” añadimos estas dos líneas de código para que se resten los puntos:

```
driver.setPoints(driver.getPoints()-points);
dao.update(driver);
```

- En la clase “Inquiry”, métodos “calculatePoints” y “calculateAmounts” añadimos el código necesario para contemplar el caso de velocidad máxima 50, pues falta en ambos casos:

```
} else if (maxSpeed==50) {
                    if (speed>=51 && speed<=70)
                           return 0;
                    else if (speed>=71 && speed<=80)
                           return 2;
                    else if (speed>=81 && speed<=90)
                           return 4;
                    else if (speed>=91)
                           return 6;
```

```
} else if (maxSpeed==50) {
                    if (speed>=51 && speed<=70)
                           return 100;
                    else if (speed>=71 && speed<=80)
                           return 300;
                    else if (speed>=81 && speed<=90)
                           return 400;
                    else if (speed>=91 && speed<=100)
                           return 500;
                    else
return 600;
```

- En la clase “Inquiry”, método “calculateAmounts” corregimos un error existente en una de las cantidades de la multa para velocidad máxima 30, pone “6” cuando debería poner “600”:

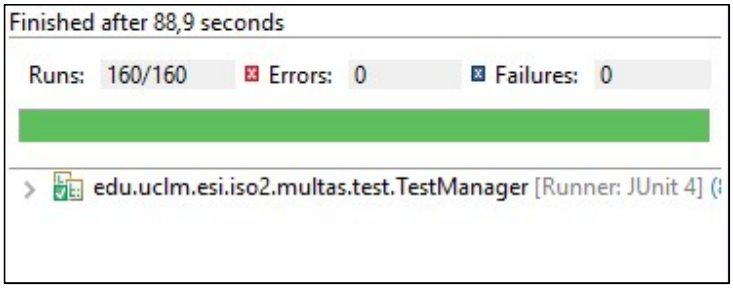
```
if (maxSpeed==30) {
                    if (speed>=31 && speed<=50)
                           return 100;
                    else if (speed>=51 && speed<=60)
                           return 300;
                    else if (speed>=61 && speed<=70)
                           return 400;
                    else if (speed>=71 && speed<=80)
                           return 500;
                    else
      return 600;
```

- Para controlar el caso de que lleguen sanciones por circular a velocidades “legales”, hecho que no debería suceder pero que se decide controlar por si ocurre, modificamos el método “openInquiry” de la clase “Manager” de tal forma que lanza una excepción si llega una multa cuya *speed* es menor o igual que *maxSpeed*. Así quedaría el método después de la modificación:

```
      public int openInquiry(String license, double speed, String location,
double maxSpeed) throws Exception {
            if(speed<=maxSpeed)
                  throw new Exception();
            Inquiry inquiry=new Inquiry(license, speed, location, maxSpeed);
            GeneralDao<Inquiry> dao=new GeneralDao<>();
            dao.insert(inquiry);
            return inquiry.getId();
}
```

C) RESULTADOS TRAS LAS MODIFICACIONES

Tras implementar todas las modificaciones descritas anteriormente, volvemos a ejecutar todos los casos de prueba, siendo ya todos ellos correctos



D) REFACTORIZACIONES

Comprobamos que son susceptibles de ser refactorizados los métodos “calculateAmounts” y “calculatePoints” de la clase “Inquiry”. Tras el proceso de refactorización, quedan de la siguiente manera:

```
	private int calculatePoints() {
		double diferencia = speed - maxSpeed;

		if (maxSpeed >= 30 && maxSpeed <= 50) {
			if (diferencia <= 20) {
				return 0;
			} else {
				if (diferencia <= 30) {
					return 2;
				} else {
					if (diferencia <= 40) {
						return 4;
					} else {
						return 6;
					}
				}
			}
		} else {
			if (diferencia <= 30) {
				return 0;
			} else {
				if (diferencia <= 50) {
					return 2;
				} else {
					if (diferencia <= 60) {
						return 4;
					} else {
						return 6;
					}
				}
			}
		}
	}
```

```
	private int calculateAmount() {
		double diferencia = speed - maxSpeed;

		if (maxSpeed >= 30 && maxSpeed <= 50) {
			if (diferencia <= 20) {
				return 100;
			} else {
				if (diferencia <= 30) {
					return 300;
				} else {
					if (diferencia <= 40) {
						return 400;
					} else {
						if (diferencia <= 50) {
							return 500;
						} else {
							return 600;
						}
					}
				}
			}
		} else {
			if (diferencia <= 30) {
				return 100;
			} else {
				if (diferencia <= 50) {
					return 300;
				} else {
					if (diferencia <= 60) {
						return 400;
					} else {
						if (diferencia <= 70) {
							return 500;
						} else {
							return 600;
						}
					}
				}
			}
		}
	}
}
```

Tras el proceso, pasamos de un total de 207 líneas para los dos métodos a 74 líneas, lo que supone una reducción del 64%, además de la mejora en claridad y legibilidad del código que se ha conseguido.

E) RESULTADO TRAS LAS REFACTORIZACIONES

Una vez realizada la refactorización, volvemos a ejecutar todos los casos de prueba, comprobando que todos ellos siguen dando un resultado positivo: