```
package domain;

import static org.junit.Assert.*;

import org.junit.Before;
import org.junit.Test;

public class CirkelTest {
	Punt punt = new Punt(0,0);
	Cirkel cirkel = new Cirkel(punt,5);
	Omhullende o;

	@Before
	public void setUp(){
		o= new Omhullende(20,20,new Punt(90,90));
	}

	@Test
	public void test_consturctor(){
		Cirkel c= new Cirkel(new Punt(100,100), 10);
		assertEquals(10, c.getRadius());
		assertEquals(100, c.getMiddelPunt().getX());
		assertEquals(100, c.getMiddelPunt().getY());
		assertEquals(o, c.getOmhullende());

	}
	@Test
```

```
	public void test_Maakt_Geldig_MiddelPunt_En_Straal_Aan(){
		assertEquals(cirkel.getMiddelPunt(), punt);
		assertEquals(cirkel.getRadius(), 5);
	}

	@Test (expected = IllegalArgumentException.class)
	public void test_Middelpunt_Null_Geeft_Exeption(){
		cirkel.setMiddelPunt(null);
	}

	@Test(expected = IllegalArgumentException.class)
	public void test_Negatieve_Straal_Gooit_Exeption(){
		cirkel.setRadius(-5);
	}

	@Test(expected = IllegalArgumentException.class)
	public void test_Straal_Nul_Gooit_Exeption(){
		cirkel.setRadius(0);
	}
	@Test
	public void test_toString_Werkt_Correct(){
		String expected = "Cirkel: middelPunt: "+punt.toString()+" - straal:
"+cirkel.getRadius()+"\n"+cirkel.getOmhullende().toString();
		assertEquals(expected, cirkel.toString());
	}

	@Test
	public void test_Equals_Twee_Gelijke_Cirkels_Werkt(){
```

```
        assertTrue(cirkel.equals(cirkel));
}

@Test
public void test_Equals_Faalt_Wanneer_Cirkel_null(){
        assertFalse(cirkel.equals(null));
}



@Test
public void test_Equals_Twee_Cirkels_Met_Verschillend_MiddelPunt_Geeft_false(){
        Punt punt1 = new Punt(1,1);
        Cirkel cirkel1 = new Cirkel(punt1,5);

        assertFalse(cirkel.equals(cirkel1));
}

@Test
public void test_Equals_Twee_Cirkels_Met_Verschillende_Straal_Geeft_false(){

        Cirkel cirkel1 = new Cirkel(punt,10);

        assertFalse(cirkel.equals(cirkel1));
}
```

```
}
package domain;

import static org.junit.Assert.*;

import org.junit.Before;
import org.junit.Test;

public class DriehoekTest {

	private Punt hoekPunt1;
	private Punt hoekPunt2;
	private Punt hoekPunt3;
	private Driehoek driehoek;
	private Omhullende o;

	@Before
	public void setUp() throws Exception {
		hoekPunt1 = new Punt(1,1);
		hoekPunt2 = new Punt(5,10);
		hoekPunt3 = new Punt(10,5);
		driehoek = new Driehoek(hoekPunt1, hoekPunt2, hoekPunt3);
		o= new Omhullende(9,9,hoekPunt1);
	}
	@Test
	public void test_Driehoek_Aanmaken_Met_Geldig_Hoekpunten_Werkt(){
```

```
            driehoek = new Driehoek(hoekPunt1, hoekPunt2, hoekPunt3);
            assertEquals(hoekPunt1, driehoek.getHoekPunt1());
            assertEquals(hoekPunt2, driehoek.getHoekPunt2());
            assertEquals(hoekPunt3, driehoek.getHoekPunt3());
            assertEquals(o, driehoek.getOmhullende());
        }
        @Test(expected = IllegalArgumentException.class)
        public void test_Driehoek_Aanmaken_Met_Hoekpunt1_Null_Gooit_Exception(){
            hoekPunt1 = null;
            driehoek = new Driehoek(hoekPunt1, hoekPunt2, hoekPunt3);
        }
        @Test(expected = IllegalArgumentException.class)
        public void test_Driehoek_Aanmaken_Met_Hoekpunt2_Null_Gooit_Exception(){
            hoekPunt2 = null;
            driehoek = new Driehoek(hoekPunt1, hoekPunt2, hoekPunt3);
        }
        @Test(expected = IllegalArgumentException.class)
        public void test_Driehoek_Aanmaken_Met_Hoekpunt3_Null_Gooit_Exception(){
            hoekPunt3 = null;
            driehoek = new Driehoek(hoekPunt1, hoekPunt2, hoekPunt3);
        }
        @Test(expected = IllegalArgumentException.class)
        public void
test_Driehoek_Aanmaken_Met_Hoekpunt1_Gelijk_Aan_Hoekpunt2_Gooit_Exception(){
            hoekPunt1 = hoekPunt2;
            driehoek = new Driehoek(hoekPunt1, hoekPunt2, hoekPunt3);
        }
        @Test(expected = IllegalArgumentException.class)
```

```
	public void test_Driehoek_Aanmaken_Met_Hoekpunt2_Gelijk_Aan_Hoekpunt3_Gooit_Exception(){
		hoekPunt2 = hoekPunt3;
		driehoek = new Driehoek(hoekPunt1, hoekPunt2, hoekPunt3);
	}
	@Test(expected = IllegalArgumentException.class)
	public void test_Driehoek_Aanmaken_Met_Hoekpunt1_Gelijk_Aan_Hoekpunt3_Gooit_Exception(){
		hoekPunt1 = hoekPunt3;
		driehoek = new Driehoek(hoekPunt1, hoekPunt2, hoekPunt3);
	}
	@Test(expected = IllegalArgumentException.class)
	public void test_Driehoek_Aanmaken_Met_Hoekpunten_Op_1_Lijn_Gooit_Exception(){
		hoekPunt1 = new Punt(100,100);
		hoekPunt2 = new Punt(200,200);
		hoekPunt3 = new Punt(300,300);
		driehoek = new Driehoek(hoekPunt1, hoekPunt2, hoekPunt3);
	}
	@Test
	public void test_Driehoek_ToString_Werkt_Correct(){
		String expected = "Driehoek: hoekpunt1: " + driehoek.getHoekPunt1().toString() + " - hoekpunt2: " + driehoek.getHoekPunt2().toString() + " - hoekpunt3: " + driehoek.getHoekPunt3().toString()+"\n"+driehoek.getOmhullende().toString();
		assertEquals(expected, driehoek.toString());
	}
	@Test
	public void test_Driehoek_Zelfde_Hoekpunten_Is_Gelijk(){
		Driehoek driehoek2 = new Driehoek(hoekPunt1, hoekPunt2, hoekPunt3);
		assertTrue(driehoek2.equals(driehoek));
```

```
}
@Test
public void test_Driehoek_Zelfde_Hoekpunten_In_Andere_Volgorde_Is_Gelijk(){
        Driehoek driehoek2 = new Driehoek(hoekPunt2, hoekPunt3, hoekPunt1);
        assertTrue(driehoek.equals(driehoek2));

}
@Test
public void test_Driehoek_Verschillend_Als_Driehoek2_Null_Is(){
        assertFalse(driehoek.equals(null));
}
@Test
public void test_Driehoek_Verschillend_Wanneer_HoekPunt1_Verschillend(){
        hoekPunt1 = new Punt(20,0);
        Driehoek driehoek2 = new Driehoek(hoekPunt1, hoekPunt2, hoekPunt3);
        assertFalse(driehoek2.equals(driehoek));
}
@Test
public void test_Driehoek_Verschillend_Wanneer_HoekPunt2_Verschillend(){
        hoekPunt2 = new Punt(20,0);
        Driehoek driehoek2 = new Driehoek(hoekPunt1, hoekPunt2, hoekPunt3);
        assertFalse(driehoek2.equals(driehoek));
}
@Test
public void test_Driehoek_Verschillend_Wanneer_HoekPunt3_Verschillend(){
        hoekPunt3 = new Punt(20,0);
```

```
                Driehoek driehoek2 = new Driehoek(hoekPunt1, hoekPunt2, hoekPunt3);
                assertFalse(driehoek2.equals(driehoek));
        }
}
package domain;

import static org.junit.Assert.*;

import org.junit.Test;

public class HangManTest {



        @Test
        public void test_Constructor_Met_Geldige_Waardes_Werkt() {
                Speler speler= new Speler("Cath");
                WoordenLijst woorden = new WoordenLijst();
                woorden.voegToe("eb");
                TekeningHangMan tekening = new TekeningHangMan("nieuw");
                HangMan hangman = new HangMan(speler, woorden);
                assertEquals(speler, hangman.getSpeler());
                assertEquals(woorden, hangman.getWoorden());
                assertFalse(hangman.isGameover());
                assertFalse(hangman.isGewonnen());
                assertTrue(hangman.getTekening().getVorm(3).isZichtbaar());
                assertFalse(hangman.getTekening().getVorm(4).isZichtbaar());
```

```
        }
@Test(expected= IllegalArgumentException.class)
public void test_Constructor_Niet_Aangemaakt_Bij_Ongeldige_Speler(){
        WoordenLijst woorden = new WoordenLijst();
        HangMan hangman = new HangMan(null, woorden);

}
@Test(expected= IllegalArgumentException.class)
public void test_Constructor_Niet_Aangemaakt_Bij_Ongeldige_Woordenlijst(){
        Speler speler= new Speler("Cath");
        HangMan hangman = new HangMan(speler, null);

}
@Test
public void test_Fout_Raden_Tekent_Volgende_Vorm(){
        Speler speler= new Speler("Cath");
        WoordenLijst woorden = new WoordenLijst();
        woorden.voegToe("e");
        TekeningHangMan tekening = new TekeningHangMan("nieuw");
        HangMan hangman = new HangMan(speler, woorden);
        hangman.raad("z");
        hangman.raad("x");
        tekening.zetVolgendeZichtbaar();
        tekening.zetVolgendeZichtbaar();
        assertTrue(hangman.getTekening().getVorm(5).isZichtbaar());
        assertFalse(hangman.getTekening().getVorm(6).isZichtbaar());
```

```
}

@Test
public void test_14_Keer_Fout_Raden_Is_Gameover(){
	Speler speler= new Speler("Cath");
	WoordenLijst woorden = new WoordenLijst();
	woorden.voegToe("e");
	HangMan hangman = new HangMan(speler, woorden);
	for(int i =0; i < 14; i++){
		hangman.raad("z");
	}
	assertTrue(hangman.isGameover());

}
@Test
public void test_Juist_Raden_Geeft_Aangepaste_Hint(){
	Speler speler= new Speler("Cath");
	WoordenLijst woorden = new WoordenLijst();
	woorden.voegToe("e");
	HangMan hangman = new HangMan(speler, woorden);

	hangman.raad("e");
	assertEquals("e ", hangman.geefHint());
}
@Test
public void test_Alles_Juist(){
	Speler speler= new Speler("Cath");
```

```
		WoordenLijst woorden = new WoordenLijst();

		woorden.voegToe("hallo");

		HangMan hangman = new HangMan(speler, woorden);

		hangman.raad("h");

		hangman.raad("a");

		hangman.raad("l");

		hangman.raad("o");

		hangman.setGewonnen(true);

		assertEquals(true,hangman.isGewonnen());

	}

}
```

```
package domain;

import static org.junit.Assert.*;

import org.junit.Test;

public class HintTest {

	@Test
	public void test_Contructor_Geldig_woord_werkt(){

		Hint hint = new Hint("woord");

		assertEquals("woord",hint.getWoord());
	}

	@Test (expected = IllegalArgumentException.class )
	public void test_Contructor_Null_Gooit_Exeption(){
		Hint hint = new Hint(null);
	}

	@Test (expected = IllegalArgumentException.class )
	public void test_Contructor_Empty_Gooit_Exeption(){
		Hint hint = new Hint("");
	}

	@Test
	public void test_ToString_Werkt(){

		Hint hint = new Hint("woord");

		assertEquals("_ _ _ _ _", hint.toString());

	}

	@Test (expected = IllegalArgumentException.class )
	public void test_Raad_Met_Null_faalt(){
```

```
		Hint hint= new Hint("woord");
		hint.raad(null);

	}
	@Test (expected = IllegalArgumentException.class )
	public void test_Raad_Met_Lege_String_faalt(){
		Hint hint= new Hint("woord");
		hint.raad("");

	}
	@Test (expected = IllegalArgumentException.class )
	public void test_Raad_Met_Meer_Dan_1_char_faalt(){
		Hint hint= new Hint("woord");
		hint.raad("wo");

	}

	@Test
	public void test_Raad_Met_correcte_char_werkt(){
		Hint hint= new Hint("woord");

		assertTrue(hint.raad("w"));
		assertEquals("w _ _ _ _ ", hint.getVerborgenWoord());


	}
	@Test
	public void test_Raad_Met_correcte_char_werkt2(){
		Hint hint= new Hint("woord");

		assertTrue(hint.raad("d"));
		assertEquals("_ _ _ _ d ", hint.getVerborgenWoord());


	}
	@Test
	public void test_Raad_Met_foute_char_geeft_false(){
		Hint hint= new Hint("woord");

		assertFalse(hint.raad("x"));


	}

	@Test
	public void test_Raad_Met_correcte_char_werkt__en_vervangt(){
		Hint hint= new Hint("woord");
		hint.raad("w");
		assertEquals("w _ _ _ _ ",hint.getVerborgenWoord());

	}

	@Test
	public void test_equals_werkt_met_dezelde_naam(){
		Hint hint= new Hint("woord");
		Hint hint2= new Hint("woord");
		assertTrue(hint.equals(hint2));
		assertTrue(hint2.equals(hint));

	}
```

```
	@Test
	public void test_equals_werkt_met_verschillende_naam(){
		Hint hint= new Hint("woord");
		Hint hint2= new Hint("woordje");
		assertFalse(hint.equals(hint2));

	}
	@Test(expected = IllegalArgumentException.class )
	public void test_equals_werkt_met_lege_naam(){
		Hint hint= new Hint("woord");
		Hint hint2= new Hint("");


	}

	@Test(expected = IllegalArgumentException.class )
	public void test_equals_werkt_met_null_naam(){
		Hint hint= new Hint("woord");
		Hint hint2= new Hint(null);

	}


}
```

```
package domain;


import static org.junit.Assert.*;


import org.junit.Before;

import org.junit.Test;


public class LijnstukTest {

	Lijnstuk lijnstuk,lijnstuk2,lijnstuk3;


	Punt punt, punt2, punt3;

	Omhullende o;


	@Before

	public void setup() {
```

```
        punt = new Punt(100, 200);
        punt2 = new Punt(100, 300);

        o= new Omhullende(0, 100, punt);


        lijnstuk = new Lijnstuk(punt, punt2);
        lijnstuk2 = new Lijnstuk(punt, punt2);

}

@Test
public void testConstructor_Geldige_StartPunt_EindPunt() {
        Lijnstuk lijnstuk=new Lijnstuk(punt, punt2);
        assertEquals(punt, lijnstuk.getStartPunt());
        assertEquals(punt2, lijnstuk.getEindPunt());
        assertEquals(o, lijnstuk.getOmhullende());
}

@Test
public void testSetter_Geldige_StartPunt_EindPunt() {
        lijnstuk.setStartEnEindPunt(punt, new Punt(100, 50));
        assertEquals(punt, lijnstuk.getStartPunt());
        assertEquals(new Punt(100, 50), lijnstuk.getEindPunt());
}

@Test(expected = IllegalArgumentException.class)
```

```
	public void Setter_Met_Ongeldige_Start_Eind_Punt() {
		lijnstuk.setStartEnEindPunt(null, null);
	}

	@Test(expected = IllegalArgumentException.class)
	public void Setter_Met_Hetzelfde_Start_Eind_Punt() {
		lijnstuk.setStartEnEindPunt(punt2, punt2);
	}

	@Test
	public void Geldige_ToString() {
		assertEquals("Lijn: startpunt: " + lijnstuk.getStartPunt()
					+ "- eindpunt" +
lijnstuk.getEindPunt()+"\n"+lijnstuk.getOmhullende().toString(), lijnstuk.toString());
	}

	@Test
	public void Geldige_Equals_Methode() {
		assertTrue(lijnstuk.equals(lijnstuk2));
		assertTrue(lijnstuk2.equals(lijnstuk));
	}

	@Test
	public void Geldige_Equals_Methode_Start_En_EindPunt_Omgewisseld() {
		Lijnstuk lijnstuk= new Lijnstuk(punt2, punt);
		assertTrue(lijnstuk.equals(lijnstuk2));
		assertTrue(lijnstuk2.equals(lijnstuk));
	}
```

```
	@Test
	public void Ongeldige_Equals_Methode_Met_Null() {

		assertFalse(lijnstuk.equals(null));
	}

	@Test
	public void Ongeldige_Equals_Methode_Met_Ander_StartPunt() {
		assertFalse(lijnstuk.equals(lijnstuk3));
	}

	@Test
	public void Ongeldige_Equals_Methode_Met_Ander_EindPunt() {
		Lijnstuk lijnstuk1 = new Lijnstuk(punt, new Punt(50,50));
		assertFalse(lijnstuk.equals(lijnstuk1));
	}
}
package domain;

import static org.junit.Assert.*;

import org.junit.Before;
import org.junit.Test;

public class OmhullendeTest {
	Punt punt1;
```

```
@Before
public void setUp(){
	punt1= new Punt(100,200);

}
@Test
public void test_construcor_werkt_met_geldige_parameters() {
	Omhullende omhullende = new Omhullende(20, 50, punt1);
	assertEquals(20,omhullende.getBreedte());
	assertEquals(50,omhullende.getHoogte());
	assertEquals(punt1,omhullende.getPositie());

}

@Test(expected=IllegalArgumentException.class)
public void test_construcor_werkt_niet_met_ongeldig_punt(){
	Omhullende omhullende = new Omhullende(20, 50, null);



}
@Test(expected=IllegalArgumentException.class)
public void test_construcor_werkt_niet_met_ongeldig_hoogte(){
	Omhullende omhullende = new Omhullende(20, -50, null);



}
```

```
	@Test(expected=IllegalArgumentException.class)
	public void test_construcor_werkt_niet_met_ongeldig_breedte(){
		Omhullende omhullende = new Omhullende(-20, 50, punt1);



	}

	@Test
	public void test_toString_werkt_met_geldige_invoer(){
		Omhullende omhullende = new Omhullende(20, 50, punt1);
		assertEquals(("Omhullende: (" + omhullende.getPositie().getX() + ", "
				+ omhullende.getPositie().getY() + ")" + " - " +
omhullende.getBreedte()
				+ " - " + omhullende.getHoogte()),omhullende.toString());



	}

	@Test
	public void test_equals_omhullende_zijn_gelijk_met_zelfde_hoogte_breetde_positie(){
		Omhullende omhullende1 = new Omhullende(20, 50, punt1);
		Omhullende omhullende2 = new Omhullende(20, 50, punt1);

		assertTrue(omhullende1.equals(omhullende2));
		assertTrue(omhullende2.equals(omhullende1));

	}
```

```
@Test
public void test_equals_omhullende_zijn_verschillend_als_omhullende_null_is(){
        Omhullende omhullende1 = new Omhullende(20, 50, punt1);
        Omhullende omhullende2 =null;

        assertFalse(omhullende1.equals(omhullende2));


}
@Test
public void test_equals_omhullende_zijn_verschillend_als_breedte_verschillend_is(){
        Omhullende omhullende1 = new Omhullende(20, 50, punt1);
        Omhullende omhullende2 =new Omhullende(30, 50, punt1);

        assertFalse(omhullende1.equals(omhullende2));


}

@Test
public void test_equals_omhullende_zijn_verschillend_als_hoogte_verschillend_is(){
        Omhullende omhullende1 = new Omhullende(20, 50, punt1);
        Omhullende omhullende2 =new Omhullende(20, 10, punt1);

        assertFalse(omhullende1.equals(omhullende2));
```

```
	}
	@Test
	public void test_equals_omhullende_zijn_verschillend_als_positie_verschillend_is(){
		Omhullende omhullende1 = new Omhullende(20, 50, punt1);
		Omhullende omhullende2 =new Omhullende(20, 50, new Punt(200,200));

		assertFalse(omhullende1.equals(omhullende2));


	}


}
package domain;

import static org.junit.Assert.*;

import org.junit.Before;
import org.junit.Test;

public class PuntTest {

	private int xCoordinaat;
	private int yCoordinaat;
	private Punt punt;
```

```
@Before
public void setUp() throws Exception {
	xCoordinaat = 5;
	yCoordinaat = 10;
	punt = new Punt(xCoordinaat, yCoordinaat);
}

@Test
public void test_Punt_wordt_correct_aangemaakt() {
	punt = new Punt(xCoordinaat, yCoordinaat);
	assertEquals(xCoordinaat, punt.getX());
	assertEquals(yCoordinaat, punt.getY());
}

@Test
public void test_toString_geeft_eigenschappen_punt(){
	String expected = "("+xCoordinaat+", "+yCoordinaat+")";
	assertEquals(expected, punt.toString());
}

@Test
public void test_equals_geeft_false_voor_parameter_null(){
	assertFalse(punt.equals(null));
}

@Test
public void test_equals_geeft_false_voor_punt_met_verschillende_xCoordinaat(){
```

```
            Punt puntAnder = new Punt(xCoordinaat-1, yCoordinaat);
            assertFalse(punt.equals(puntAnder));
        }

        @Test
        public void test_equals_geeft_false_voor_punt_met_verschillende_yCoordinaat(){
            Punt puntAnder = new Punt(xCoordinaat, yCoordinaat-1);
            assertFalse(punt.equals(puntAnder));
        }



        @Test
        public void test_equals_geeft_true_voor_punt_met_dezelfde_x_en_yCoordinaat(){
            Punt puntAnder = new Punt(xCoordinaat, yCoordinaat);
            assertTrue(punt.equals(puntAnder));
        }



}
package domain;

import static org.junit.Assert.*;

import org.junit.Before;
import org.junit.Test;

public class RechthoekTest {
```

```
Punt punt,punt2;
Rechthoek rechthoek1, rechthoek2,rechthoek3;
Omhullende o;
	@Before
	public void setUp(){
		punt= new Punt(0, 50);
		punt2= new Punt(10, 20);
		rechthoek1= new Rechthoek(80, 10, punt);
		rechthoek2=new Rechthoek(80, 10, punt);
		rechthoek3=new Rechthoek(10,10,punt);
		o=new Omhullende(80, 50, punt);

	}


	@Test
	public void test_of_contructor_juist_is() {
		Rechthoek rechthoek= new Rechthoek(50, 80, punt);
		assertEquals(50, rechthoek.getHoogte());
		assertEquals(80, rechthoek.getBreedte());
		assertEquals(punt, rechthoek.getPositieLinksBoven());
		assertEquals(o, rechthoek.getOmhullende());


	}

	@Test
```

```
public void test_of_set_hoogte_juist_is(){
        rechthoek1.setHoogte(80);
        assertEquals(80, rechthoek1.getHoogte());
}
@Test
public void test_of_set_breedte_juist_is(){
        rechthoek1.setBreedte(10);
        assertEquals(10, rechthoek1.getBreedte());
}
@Test
public void test_of_set_positie_juist_is(){
        rechthoek1.setPositieLinksBoven(punt2);
        assertEquals(punt2, rechthoek1.getPositieLinksBoven());
}

@Test(expected=IllegalArgumentException.class)
public void test_of_set_hoogte_fout_is(){
        rechthoek1.setHoogte(-20);
}



@Test(expected=IllegalArgumentException.class)
public void test_of_set_hoogte_fout_is_met_waarde_0(){
        rechthoek1.setHoogte(0);
}
```

```
@Test(expected=IllegalArgumentException.class)
public void test_of_set_breedte_fout_is(){
        rechthoek1.setBreedte(-100);

}

@Test(expected=IllegalArgumentException.class)
public void test_of_set_breedte_fout_is_met_waarde0(){
        rechthoek1.setBreedte(0);

}
@Test(expected=IllegalArgumentException.class)
public void test_of_set_positie_fout_is(){

        rechthoek1.setPositieLinksBoven(null);
}
@Test
public void Test_of_equals_methode_juist_werkt(){
        assertTrue(rechthoek1.equals(rechthoek2));
        assertTrue(rechthoek2.equals(rechthoek1));

}
@Test(expected=IllegalArgumentException.class)
public void Test_equals_geeft_false_als_positie_null_is(){
        Rechthoek rechthoek2=new Rechthoek(10,10,null);
        assertFalse(rechthoek1.equals(rechthoek2));
```

```
	}

	@Test
	public void Test_equals_geeft_false_als_rechthoek_positie_verschillend_is(){
		Rechthoek rechthoek2=new Rechthoek(10,10,punt2);
		assertFalse(rechthoek1.equals(rechthoek2));

	}
	@Test
	public void Test_equals_geeft_false_als_rechthoek_hoogte_verschillend_is(){
		Rechthoek rechthoek2=new Rechthoek(10,10,punt);
		assertFalse(rechthoek1.equals(rechthoek2));

	}

	@Test
	public void Test_equals_geeft_false_als_rechthoek_breedte_verschillend_is(){
		Rechthoek rechthoek2=new Rechthoek(80,20,punt);
		assertFalse(rechthoek1.equals(rechthoek2));

	}
	@Test
	public void Test_of_toString_klopt(){
			assertEquals("Rechthoek: positie: "+
rechthoek1.getPositieLinksBoven().toString()+" - "+ " breedte: "+ rechthoek1.getBreedte()+" -
"+"Hoogte: "+rechthoek1.getHoogte()+"\n"+rechthoek1.getOmhullende(), rechthoek1.toString());
	}
```

```
}
package domain;

import static org.junit.Assert.*;

import org.junit.Before;
import org.junit.Test;

public class SpelerTest {

	private String naam;
	private String anderenaam;
	private int scoreToAdd;
	private int wrongscoreToAdd;
	private Speler speler;

	@Before
	public void setUp() throws Exception {
		naam = "Lars";
		anderenaam = "Lies";
		scoreToAdd = 5;
		wrongscoreToAdd = -5;
		speler = new Speler(naam);
	}

	@Test
	public void test_Speler_wordt_correct_aangemaakt() {
```

```
        speler = new Speler(naam);
        assertEquals(naam, speler.getNaam());
        assertEquals(0, speler.getScore());
}

@Test (expected = IllegalArgumentException.class)
public void test_Speler_Exception_bij_parameter_null() {
        speler = new Speler(null);
}

@Test (expected = IllegalArgumentException.class)
public void test_Speler_Exception_bij_parameter_lege_string() {
        speler = new Speler("");
}

@Test
public void test_toString_geeft_eigenschappen_speler(){
        String expected = naam + ": " + 0;
        assertEquals(expected, speler.toString());
}

@Test
public void test_equals_geeft_false_voor_parameter_null(){
        assertFalse(speler.equals(null));
}

@Test
```

```
	public void test_equals_geeft_false_voor_speler_met_andere_naam(){
		Speler andereSpeler = new Speler(anderenaam);
		assertFalse(speler.equals(andereSpeler));
	}

	@Test
	public void test_equals_geeft_false_voor_speler_met_andere_score(){
		Speler andereSpeler = new Speler(naam);
		andereSpeler.addToScore(scoreToAdd);
		assertFalse(speler.equals(andereSpeler));
	}

	@Test
	public void test_addtoScore_voegt_score_correct_toe_voor_positieve_score(){
		speler.addToScore(scoreToAdd);
		assertEquals(scoreToAdd, speler.getScore());
	}

	@Test
	public void
test_addtoScore_voegt_score_correct_toe_voor_negatieve_score_kleiner_dan_huidige_score(){
		speler.addToScore(scoreToAdd);
		speler.addToScore(scoreToAdd);
		speler.addToScore(wrongscoreToAdd);
		assertEquals(scoreToAdd, speler.getScore());
	}

	@Test (expected = IllegalArgumentException.class)
```

```
	public void test_addtoScore_Exception_als_resulterende_score_negatief(){
		speler.addToScore(wrongscoreToAdd);
	}

}
package domain;

import static org.junit.Assert.*;

import org.junit.Before;
import org.junit.Test;

import ui.HangmanPanel;

public class TekeningHangMantest {
	TekeningHangMan tekening;
	Punt punt = new Punt(0,0);
	Cirkel cirkel = new Cirkel(punt,5);

	@Before
	public void setUp(){
		tekening = new TekeningHangMan("blabla");
	}
	@Test
	public void test_Naam_Is_Correct() {
		assertEquals(tekening.getNaam(), "Hangman");
```

```
}

@Test
public void test_Tekening_Bevat_18Vormen(){
        assertEquals(tekening.getAantalVormen(),18);
}

@Test
public void test_Eerste4_Vormen_Zijn_Zichtbaar(){
        assertTrue(tekening.getVorm(0).isZichtbaar());
        assertTrue(tekening.getVorm(1).isZichtbaar());
        assertTrue(tekening.getVorm(2).isZichtbaar());
        assertTrue(tekening.getVorm(3).isZichtbaar());
}

@Test
public void test_De_Laatste_14_Vormen_Zijn_Onzichtbaar(){

        System.out.println(tekening.getAantalVormen());


        assertFalse(tekening.getVorm(4).isZichtbaar());
        assertFalse(tekening.getVorm(5).isZichtbaar());
        assertFalse(tekening.getVorm(6).isZichtbaar());
        assertFalse(tekening.getVorm(7).isZichtbaar());
        assertFalse(tekening.getVorm(8).isZichtbaar());
        assertFalse(tekening.getVorm(9).isZichtbaar());
```

```
                assertFalse(tekening.getVorm(10).isZichtbaar());
                assertFalse(tekening.getVorm(11).isZichtbaar());
                assertFalse(tekening.getVorm(12).isZichtbaar());
                assertFalse(tekening.getVorm(13).isZichtbaar());
                assertFalse(tekening.getVorm(14).isZichtbaar());
                assertFalse(tekening.getVorm(15).isZichtbaar());
                assertFalse(tekening.getVorm(16).isZichtbaar());
                assertFalse(tekening.getVorm(17).isZichtbaar());
        }

        @Test (expected = IllegalArgumentException.class)
        public void test_Voegtoe_Gooit_Een_Exception(){
                tekening.voegToe(cirkel);
        }

        @Test (expected = IllegalArgumentException.class)
        public void test_Verwijder_Gooit_Een_Exception(){
                tekening.verwijder(cirkel);
        }
}
package domain;

import static org.junit.Assert.assertEquals;
import static org.junit.Assert.assertFalse;
import static org.junit.Assert.assertTrue;

import org.junit.Before;
```

```
import org.junit.Test;


public class TekeningTest {
	Punt punt;
	Rechthoek rechthoek1,rechthoek2;
	@Before
	public void setup(){
		punt= new Punt(0, 50);


		rechthoek1 = new Rechthoek(80, 10, punt);
		rechthoek2= new Rechthoek(80, 80, punt);
	}


	@Test
	public void testConstructor_kan_geldige_tekeing_aanmaken() {
		Tekening tekening= new Tekening("tekening");
		assertEquals("tekening", tekening.getNaam());
		assertEquals(0, tekening.getAantalVormen());



	}
	@Test (expected = IllegalArgumentException.class)
	public void testConstructor_faalt_wanneer_naam_is_null() {
		Tekening tekening= new Tekening(null);


	}
	@Test (expected = IllegalArgumentException.class)
```

```
public void testConstructor_faalt_wanneer_naam_is_leeg() {
        Tekening tekening= new Tekening("");

}
@Test
public void testVormGrootte_kan_Nul_zijn(){
        Tekening tekening = new Tekening("ge");

        assertEquals(0,tekening.getAantalVormen());
}

@Test
public void testAantalVormen_geeft_juiste_aantal_terug(){
        Tekening tekening= new Tekening("tekening");
        tekening.voegToe(rechthoek1);
        assertEquals(1,tekening.getAantalVormen());

}

@Test
public void testVoegToe_voegt_een_vorm_toe(){
        Tekening tekening= new Tekening("tekening");
        tekening.voegToe(rechthoek1);
        assertEquals(1,tekening.getAantalVormen());
}
@Test (expected = IllegalArgumentException.class)
public void testVoegToe_gooitException_als_vorm_al_bestaat(){
```

```
        Tekening tekening= new Tekening("tekening");
        tekening.voegToe(rechthoek1);
        tekening.voegToe(rechthoek1);

}
@Test (expected = IllegalArgumentException.class)
public void testVoegToe_gooitException_als_vorm_null_is(){
        Tekening tekening= new Tekening("tekening");
        tekening.voegToe(null);
}
@Test
public void testgetVorm_geeft_juiste_index_terug(){

        Tekening tekening= new Tekening("tekening");
        tekening.voegToe(rechthoek1);

        tekening.voegToe(rechthoek2);
        assertEquals(rechthoek2,tekening.getVorm(1));
}
@Test(expected = IllegalArgumentException.class)
public void testgetVorm_gooit_exception_bij_negatieve_index(){
        Tekening tekening= new Tekening("tekening");
        tekening.voegToe(rechthoek1);
        tekening.voegToe(rechthoek2);
        assertEquals(rechthoek2,tekening.getVorm(-1));
}
@Test(expected = IllegalArgumentException.class)
```

```
public void testgetVorm_gooit_exception_bij_te_grootte_index(){
        Tekening tekening= new Tekening("tekening");
        tekening.voegToe(rechthoek1);
        tekening.voegToe(rechthoek2);
        assertEquals(rechthoek2,tekening.getVorm(3));
}
@Test
public void testbevat_geeft_true_bij_vorm_in_tekening(){
        Tekening tekening= new Tekening("tekening");
        tekening.voegToe(rechthoek1);
        tekening.voegToe(rechthoek2);
        assertTrue(tekening.bevat(rechthoek1));
}
@Test
public void testbevat_geeft_false_bij_vorm_niet_intekening(){
        Tekening tekening= new Tekening("tekening");
        tekening.voegToe(rechthoek1);

        assertFalse(tekening.bevat(rechthoek2));
}
@Test
public void testVerwijder_werkt(){
        Tekening tekening= new Tekening("tekening");
        tekening.voegToe(rechthoek1);
        tekening.verwijder(rechthoek1);
        assertEquals(0,tekening.getAantalVormen());
}
```

```
@Test
public void testverwijder_doet_niets_bij_vormisnull(){
        Tekening tekening= new Tekening("tekening");
        tekening.voegToe(rechthoek1);
        tekening.verwijder(null);
        assertEquals(1,tekening.getAantalVormen());
}
@Test
public void testVerwijder_doet_niets_als_vorm_niet_in_tekeing_is(){
        Tekening tekening= new Tekening("tekening");
        tekening.voegToe(rechthoek1);
        tekening.verwijder(rechthoek2);
        assertEquals(1,tekening.getAantalVormen());

}

@Test
public void testEquals_werkt_true(){
        Tekening tekening= new Tekening("tekening");
        Tekening tekening2= new Tekening("tekening");

        assertTrue(tekening.equals(tekening2));
        assertTrue(tekening2.equals(tekening));

}
@Test
public void testEquals_geeft_false_bij_null(){
```

```
        Tekening tekening= new Tekening("tekening");

        assertFalse(tekening.equals(null));

    }
    @Test
    public void testEquals_geeft_false_bij_verschillende_naam(){
        Tekening tekening= new Tekening("tekening");
        Tekening tekening2= new Tekening("tek");

        assertFalse(tekening.equals(tekening2));
    }
    @Test
    public void testEquals_geeft_false_bij_verschillend_aantal_vormen(){

        Tekening tekening= new Tekening("tekening");
        Tekening tekening2= new Tekening("tekening");
        tekening.voegToe(rechthoek1);

        assertFalse(tekening.equals(tekening2));
    }
    @Test
    public void
testEquals_geeft_false_bij_hetzelfde_naam_zelfde_aantal_vormen_maar_1_vorm_verschillend(){

        Tekening tekening= new Tekening("tekening");
        Tekening tekening2= new Tekening("tekening");
        tekening.voegToe(rechthoek1);
```

```
        tekening2.voegToe(rechthoek2);


        assertFalse(tekening.equals(tekening2));


    }
    @Test (expected = IllegalArgumentException.class)
    public void
Test_voegToe_Vorm_gooit_exception_cirkel_met_Omhullende_met_foute_minX(){
        Tekening tekening= new Tekening("tekening");


        tekening.voegToe(new Cirkel(new Punt(10,10),50));
    }


    @Test (expected = IllegalArgumentException.class)
    public void
Test_voegToe_Vorm_gooit_exception_cirkel_met_Omhullende_met_foute_minY(){
        Tekening tekening= new Tekening("tekening");


        tekening.voegToe(new Cirkel(new Punt(10,5),50));
    }


    @Test (expected = IllegalArgumentException.class)
    public void
Test_voegToe_Vorm_gooit_exception_cirkel_met_Omhullende_met_foute_maxX(){
        Tekening tekening= new Tekening("tekening");


        tekening.voegToe(new Cirkel(new Punt(220,10),50));
    }
```

```
	@Test (expected = IllegalArgumentException.class)
	public void
Test_voegToe_Vorm_gooit_exception_cirkel_met_Omhullende_met_foute_maxY(){
		Tekening tekening= new Tekening("tekening");

		tekening.voegToe(new Cirkel(new Punt(200,390),50));
	}
}
package domain;

import static org.junit.Assert.*;

import org.junit.Test;

public class WoordenLijstTest {



	@Test
	public void test_constructot_wordt_geinitialiseerd_met_lege_woordenlijst() {
		WoordenLijst w=  new WoordenLijst();
		assertEquals(0, w.getAantalWoorden());

	}

	@Test
	public void test_voegtoe_met_geldige_woord() {
```

```
        WoordenLijst w=  new WoordenLijst();
        w.voegToe("woord");
        assertEquals(1, w.getAantalWoorden());

}
@Test(expected=IllegalArgumentException.class)
public void test_voegtoe_met_lege_waarde_gooit_exception() {
        WoordenLijst w=  new WoordenLijst();
        w.voegToe("");

}
@Test(expected=IllegalArgumentException.class)
public void test_voegtoe_met_null_waarde_gooit_exception() {
        WoordenLijst w=  new WoordenLijst();
        w.voegToe(null);
}

@Test(expected=IllegalArgumentException.class)
public void test_voegtoe_met_dubbel_voorkomen_gooit_exception() {
        WoordenLijst w=  new WoordenLijst();
        w.voegToe("woord");
        w.voegToe("woord");

}


@Test
```

```
public void test_toString_geeft_lijst_met_woorden() {
        WoordenLijst w=  new WoordenLijst();
        w.voegToe("woord");

        assertEquals("woord\n", w.toString());
}


@Test(expected=IllegalArgumentException.class)
public void test_geef_random_woord_gooit_exception_bij_lege_lijst() {
        WoordenLijst w=  new WoordenLijst();
        w.geeftRandomWoord();
}
@Test
public void test_geef_random_woord_werkt_bij_1_woord() {
        WoordenLijst w=  new WoordenLijst();
        w.voegToe("woord");
        assertEquals("woord", w.geeftRandomWoord());

}

@Test
public void test_geef_random_woord_werkt_bij_meerdere_woorden() {
        WoordenLijst w=  new WoordenLijst();
        w.voegToe("woord");
        w.voegToe("woorden");
        w.voegToe("woordje");
```

```
            String woord= w.geeftRandomWoord();

            assertTrue(w.bevat(woord));

        }

}
```