Fakultet elektrotehnike, računarstva i informacijskih znanosti Osijek

Vježba 4: Pouzdanost i testiranje programske podrške

Marko Dubravac DRC – LV2

17.1.2024.

1. Zadatak

Stroj za kavu mora imati varijable kao što su: količina kave, količina vode, status stroja – definirana svojstva za kavu i vodu moraju biti numeričkog tipa, dok status stroja mora biti Boolean tipa. Stroj za kavu mora imati metode za paljenje i gašenje stroja. Svaka od navedenih metoda za paljenje i gašenje mora vratiti status Boolean tipa. Ako je stroj upaljen i ponovo ga se pali, metoda mora vratiti poruku da je stroj već upaljen. Također ako je stroj ugašen, i pokušava ga se ponovno ugasiti – potrebno je vratiti poruku da je stroj već ugašen. Osim metode za paljenje i gašenje, potrebno je definirati metodu koja nadopunjuje količinu kave i vode. Predani argumenti navedenoj metodi moraju biti numeričkog tipa – ako je nadopuna uspješna, metoda mora vratiti status 200. Ako predani argumenti nisu numeričkog tipa, metoda mora vratiti grešku s definiranom porukom. Za kraj potrebno je definirati metodu za pravljenje kave. Sve dok ima više od 5g kave i 15ml vode stroj može praviti kavu. Ako stroj ima manje od navedenih količina, mora vratiti poruku da je potrebno nadopuniti definirane količine i ne smije omogućiti pravljenje nove kave.

zad1.js

```
class CoffeMachine {
  constructor(coffeeAmount, waterAmount, powerStatus) {
    if (typeof coffeeAmount !== "number" || isNaN(coffeeAmount)) {
      throw new Error("Coffee amount must be a number");
    }
    if (typeof waterAmount !== "number" || isNaN(waterAmount)) {
      throw new Error("Water amount must be a number");
    }
    if (typeof powerStatus !== "boolean") {
      throw new Error("Power status must be a boolean");
    }
    this.coffeeAmount = coffeeAmount;
    this.waterAmount = waterAmount;
    this.powerStatus = powerStatus;
  }
  turnMachineOn() {
    if (this.powerStatus === false) {
      this.powerStatus = true;
      return this.powerStatus;
    }
    console.log("Machine is already turned on !");
    return this.powerStatus;
  }
  turnMachineOff() {
    if (this.powerStatus === true) {
      this.powerStatus = false;
      return this.powerStatus;
    }
    console.log("Machine is already turned off !");
    return this.powerStatus;
  }
  refill(coffe, water) {
    if (Number.isInteger(coffe) && Number.isInteger(water)) {
      this.coffeAmount = coffe;
```

```
      this.waterAmmount = water;
      console.log("Machine refilled !");
      return 200;
    }
    throw new Error("Illegal type");
  }
  makeCoffe() {
    if (this.powerStatus === false) {
      throw new Error("Machine is off !");
    }
    this.coffeAmount -= 5;
    this.waterAmmount -= 15;
  }
}
module.exports = CoffeMachine;
```

zad1.test.js

```
const assert = require("chai").assert;

const CoffeMachine = require("./../zad1");

const coffeAmount = 10;
const waterAmmount = 20;
const onPowerStatus = true;
const offPowerStatus = false;

const testObject = new CoffeMachine(coffeAmount, waterAmmount,
onPowerStatus);
const testObjectOff = new CoffeMachine(
  coffeAmount,
  waterAmmount,
  offPowerStatus,
);

describe("Coffe machine unit test", () => {
  describe("Test konstruktora", () => {
    it("Za ispravan CM vratiti ipravno", () => {
      let newCM = new CoffeMachine(100, 100, true);
      assert.equal(newCM, newCM);
    });
    it("Za los CM vratiti error - number is not a string", () => {
      assert.throws(() => {
        let newCM = new CoffeMachine("notnumber", 100, true);
      }, Error);
    });
    it("Za los CM vratiti error - number is not a string", () => {
      assert.throws(() => {
        let newCM = new CoffeMachine(100, "notnumber", true);
      }, Error);
    });
    it("Za los CM vratiti error - string is not boolean", () => {
      assert.throws(() => {
        let newCM = new CoffeMachine(100, 100, "hello");
      }, Error);
```

```
    });
  });
  describe("Test metode turnMachineOn(): ", () => {
    it("Za testni CM mora vratiti status true", () => {
      assert.isTrue(testObject.turnMachineOn());
    });
    it("Za ugasen CM mora vratiti status true", () => {
      let powerOffCM = new CoffeMachine(
        coffeAmount,
        waterAmmount,
        offPowerStatus,
      );
      assert.equal(powerOffCM.turnMachineOn(), true);
    });
  });
  describe("Test metode turnMachineOff(): ", () => {
    it("Za ugasen CM mora vratiti status false", () => {
      let powerOffCM = new CoffeMachine(
        coffeAmount,
        waterAmmount,
        offPowerStatus,
      );
      assert.isFalse(powerOffCM.turnMachineOff());
    });
    it("Za upaljen CM mora vratiti status false", () => {
      let powerOnCM = new CoffeMachine(
        coffeAmount,
        waterAmmount,
        onPowerStatus,
      );
      assert.equal(powerOnCM.turnMachineOff(), false);
    });
  });
  describe("Test metode refill(coffe, water): ", () => {
    it("Za ispravan CM mora vratiti status 200", () => {
      let refillCm = new CoffeMachine(coffeAmount, waterAmmount,
onPowerStatus);
      assert.equal(refillCm.refill(coffeAmount, waterAmmount), 200);
    });
    it("Za pogresan prvi parametar mora vratiti status error", () => {
      assert.throws(() => {
        assert.equal(testObject.refill("hello", waterAmmount), 200);
      }, Error);
    });
    it("Za pogresan drugi parametar mora vratiti status error", () => {
      assert.throws(() => {
        assert.equal(testObject.refill(50, "hello"), 200);
      }, Error);
    });
  });

  describe("Test metode makeCoffe(): ", () => {
    it("Za ugasen CM mora vratiti error", () => {
      assert.throws(() => {
        assert.equal(testObjectOff.makeCoffe(), 200);
      }, Error);
    });
```

```
    it("Za ispravan CM mora vratiti status 200", () => {
      let makeCoffeCM = new CoffeMachine(
        coffeAmount,
        waterAmmount,
        onPowerStatus,
      );
      assert.equal(makeCoffeCM.makeCoffe(), makeCoffeCM.makeCoffe());
    });
  });
});
```

```
Coffe machine unit test
  Test konstruktora
    ✓ Za ispravan CM vratiti ipravno
    ✓ Za los CM vratiti error - number is not a string
    ✓ Za los CM vratiti error - number is not a string
    ✓ Za los CM vratiti error - string is not boolean
  Test metode turnMachineOn():
Machine is already turned on !
    ✓ Za testni CM mora vratiti status true
    ✓ Za ugasen CM mora vratiti status true
  Test metode turnMachineOff():
Machine is already turned off !
    ✓ Za ugasen CM mora vratiti status false
    ✓ Za upaljen CM mora vratiti status false
  Test metode refill(coffe, water):
Machine refilled !
    ✓ Za ispravan CM mora vratiti status 200
    ✓ Za pogresan prvi parametar mora vratiti status error
    ✓ Za pogresan drugi parametar mora vratiti status error
  Test metode makeCoffe():
    ✓ Za ugasen CM mora vratiti error
    ✓ Za pogresan parametar mora vratiti status error
    ✓ Za pogresan parametar mora vratiti status error
    ✓ Za ispravan CM mora vratiti status 200
```

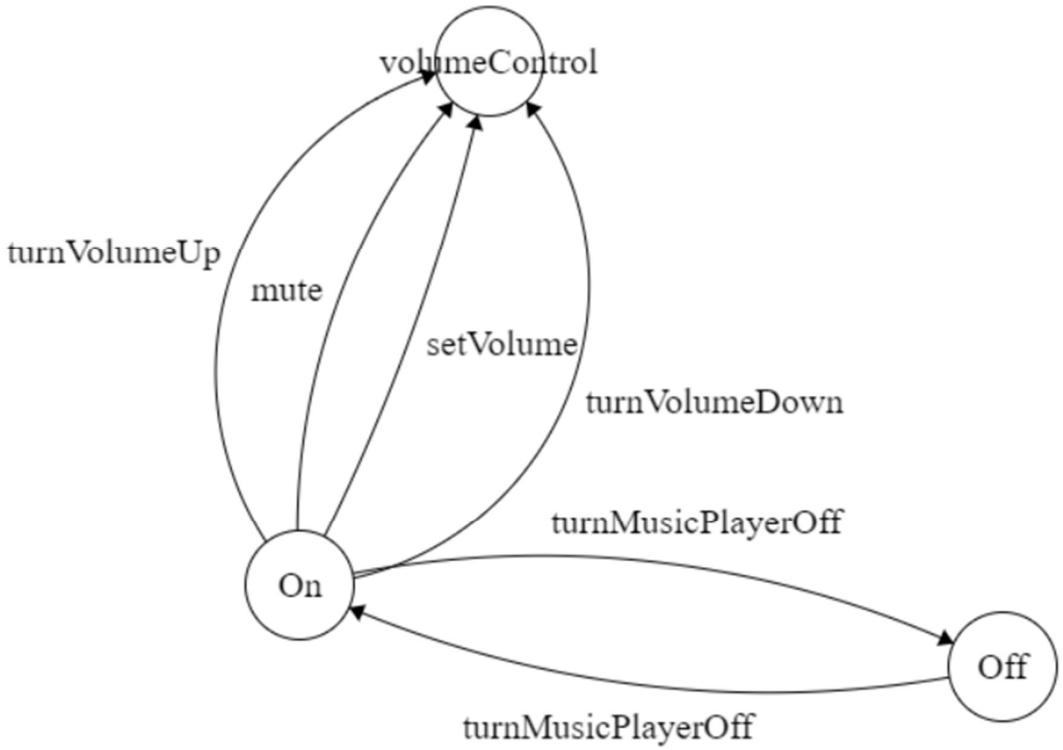
File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Line #s
All files	100	100	100	100	
zad1.js	100	100	100	100	

2. Zadatak

Definirati i implementirati zadatak (zad2.js i zad2.test.js) po vlastitom izboru navedenom TDD metodologijom.

Klasa MusicPlayer sastoji se od atributa powerStatus i volume služi. Predstavlja jednostavan zvučnik. Power Status služi za označavanje stanja: upaljen ili ugašen, a volume rezinu glasnoće na koji je uređaj postavljan. Uređaj se može upaliti i ugasiti. Uređaj se također može pojačati i stišati. Osim toga, glasnoća se može direktno postaviti i direktno ugasiti.

FSM:



zad2.js

```
class MusicPlayer {
  constructor(powerStatus, volume) {
    if (typeof powerStatus !== "boolean") {
      throw new Error("Power status must be a boolean");
    }
    if (typeof volume !== "number" || isNaN(volume)) {
      throw new Error("Music volume must be a number");
    }
    this.powerStatus = powerStatus;
    this.volume = volume;
  }
  turnMusicPlayerOn() {
    if (this.powerStatus === false) {
      this.powerStatus = true;
      return this.powerStatus;
    }
    console.log("Music player is already turned on!");
```

```
    return this.powerStatus;
  }
  turnMusicPlayerOff() {
    if (this.powerStatus === true) {
      this.powerStatus = false;
      return this.powerStatus;
    }
    console.log("Music Player is already turned off!");
    return this.powerStatus;
  }
  turnVolumeUpFor(volume) {
    if (this.powerStatus === false) return this.powerStatus;
    if (Number.isInteger(volume)) {
      this.volume = this.volume + volume;
      if (this.volume > 100) {
        console.log("Volume can't go above 100; setting it to 100");
        this.volume = 100;
        return this.volume;
      }
      console.log("Volume upped!");
      return this.volume;
    }
    throw new Error("Volume must be an integer");
  }
  turnVolumeDownFor(volume) {
    if (this.powerStatus === false) return this.powerStatus;
    if (Number.isInteger(volume)) {
      this.volume = this.volume - volume;
      if (this.volume < 0) {
        console.log("Volume can't go below 0; setting it to 0");
        this.volume = 0;
        return this.volume;
      }
      console.log("Volume downed!");
      return this.volume;
    }
    throw new Error("Volume must be an integer");
  }
  setVolume(volume) {
    if (
      Number.isInteger(volume) &&
      volume >= 0 &&
      volume <= 100 &&
      this.powerStatus === true
    ) {
      this.volume = volume;
      console.log("Volume set!");
      return this.volume;
    }
    throw new Error("Volume must be an integer between 0 and 100");
  }
  mute() {
    if (this.powerStatus === true) {
      this.volume = 0;
      console.log("Volume muted!");
      return this.volume;
    }
```

```
    throw new Error("It's already muted! Turn it on first!");
  }
}

module.exports = MusicPlayer;
```

zad2.test.js

```
const assert = require("chai").assert;

const MusicPlayer = require("./../zad2");

const volume = 10;
const onPowerStatus = true;
const offPowerStatus = false;

const testObject = new MusicPlayer(onPowerStatus, volume);
const offTestObject = new MusicPlayer(offPowerStatus, volume);

describe("Music Player unit test", () => {
  describe("Test konstruktora", () => {
    it("Za ispravan MP vratiti ipravno", () => {
      let newMP = new MusicPlayer(true, 60);
      assert.equal(newMP, newMP);
    });
    it("Za los MP vratiti error - number is not a string", () => {
      assert.throws(() => {
        let newMP = new MusicPlayer(true, "notnumber");
      }, Error);
    });
    it("Za los MP vratiti error - boolean is not a string", () => {
      assert.throws(() => {
        let newMP = new MusicPlayer("notboolean", 60);
      }, Error);
    });
  });
  describe("Test metode turnMusicPlayerOn(): ", () => {
    it("Za testni CM mora vratiti status true", () => {
      assert.isTrue(testObject.turnMusicPlayerOn());
    });
    it("Za ugasen MP mora vratiti status true", () => {
      let powerOffMP = new MusicPlayer(offPowerStatus, volume);
      assert.equal(powerOffMP.turnMusicPlayerOn(), true);
    });
  });
  describe("Test metode turnMusicPlayerOff(): ", () => {
    it("Za ugasen MP mora vratiti status false", () => {
      let powerOffMP = new MusicPlayer(offPowerStatus, volume);
      assert.isFalse(powerOffMP.turnMusicPlayerOff());
    });
    it("Za upaljen MP mora vratiti status false", () => {
      let powerOnMP = new MusicPlayer(onPowerStatus, volume);
      assert.equal(powerOnMP.turnMusicPlayerOff(), false);
    });
  });
```

```
describe("Test metode turnVolumeUpFor(volume): ", () => {
  it("Za ispravan volume mora vratiti volume", () => {
    let volumeUpMP = new MusicPlayer(onPowerStatus, 10);
    let volumeUp = volumeUpMP.volume + 10;
    assert.equal(volumeUpMP.turnVolumeUpFor(10), volumeUp);
  });
  it("Za pogresan parametar volume mora vratiti error", () => {
    assert.throws(() => {
      assert.equal(testObject.turnVolumeUpFor("hello"), 200);
    }, Error);
  });
  it("Za ugasen MP mora vratiti false", () => {
    assert.equal(offTestObject.turnVolumeUpFor(10), false);
  });
  it("Za ispravan volume preko 100 mora vratiti 100", () => {
    let volumeUpMP = new MusicPlayer(onPowerStatus, 100);
    assert.equal(volumeUpMP.turnVolumeUpFor(10), 100);
  });
});

describe("Test metode turnVolumeDownFor(volume): ", () => {
  it("Za ispravan volume mora vratiti volume", () => {
    let volumeDownMP = new MusicPlayer(onPowerStatus, 20);
    let volumeDown = volumeDownMP.volume - 10;
    assert.equal(volumeDownMP.turnVolumeDownFor(10), volumeDown);
  });
  it("Za pogresan parametar volume mora vratiti error", () => {
    assert.throws(() => {
      assert.equal(testObject.turnVolumeDownFor("hello"), 200);
    }, Error);
  });
  it("Za ugasen MP mora vratiti false", () => {
    assert.equal(offTestObject.turnVolumeDownFor(10), false);
  });
  it("Za ispravan volume ispod 0 mora vratiti 0", () => {
    let volumeDownMP = new MusicPlayer(onPowerStatus, 10);
    assert.equal(volumeDownMP.turnVolumeDownFor(20), 0);
  });
});
describe("Test metode setVolume(volume): ", () => {
  it("Za ugasen MP mora vratiti error", () => {
    assert.throws(() => {
      assert.equal(testObjectOff.setVolume(20), 200);
    }, Error);
  });
  it("Za neispravnu vrijednost argumenta mora vratiti error", () => {
    assert.throws(() => {
      assert.equal(testObjectOff.setVolume("hello"), 200);
    }, Error);
  });
  it("Za volume manji od 0 mora vratiti error", () => {
    assert.throws(() => {
      assert.equal(testObject.setVolume(-1), 200);
    }, Error);
  });
  it("Za volume veci od 100 mora vratiti error", () => {
    assert.throws(() => {
```

```
        assert.equal(testObject.setVolume(101), 200);
      }, Error);
    });
    it("Za ispravan volume mora postaviti volume", () => {
      let volumeSetMP = new MusicPlayer(onPowerStatus, 20);
      assert.equal(volumeSetMP.setVolume(60), 60);
    });
  });

  describe("Test metode mute(): ", () => {
    it("Za ugasen MP mora vratiti error", () => {
      assert.throws(() => {
        assert.equal(testObjectOff.mute(), 200);
      }, Error);
    });
    it("Za ugasen MP mora vratiti error vol2", () => {
      let offMP = new MusicPlayer(offPowerStatus, 20);
      assert.throws(() => {
        assert.equal(offMP.mute(), 200);
      }, Error);
    });
    it("Za ispravan MP treba postaviti volume na 0", () => {
      let volumeMuteMP = new MusicPlayer(onPowerStatus, 20);
      assert.equal(volumeMuteMP.mute(), 0);
    });
  });
});
```

```
      ✓ Za pogresan parametar volume mora vratiti error
      ✓ Za ugasen MP mora vratiti false
Volume can't go above 100; setting it to 100
      ✓ Za ispravan volume preko 100 mora vratiti 100
    Test metode turnVolumeDownFor(volume):
Volume downed!
      ✓ Za ispravan volume mora vratiti volume
      ✓ Za pogresan parametar volume mora vratiti error
      ✓ Za ugasen MP mora vratiti false
Volume can't go below 0; setting it to 0
      ✓ Za ispravan volume ispod 0 mora vratiti 0
    Test metode setVolume(volume):
      ✓ Za ugasen MP mora vratiti error
      ✓ Za neispravnu vrijednost argumenta mora vratiti error
      ✓ Za volume manji od 0 mora vratiti error
      ✓ Za volume veci od 100 mora vratiti error
Volume set!
      ✓ Za ispravan volume mora postaviti volume
    Test metode mute():
      ✓ Za ugasen MP mora vratiti error
Volume muted!
      ✓ Za ispravan MP treba postaviti volume na 0


  22 passing (18ms)
```

```
  23 passing (23ms)

----------|----------|----------|----------|----------|-------------------|
File      |  % Stmts | % Branch |  % Funcs |  % Lines | Uncovered Line #s |
----------|----------|----------|----------|----------|-------------------|
All files |      100 |      100 |      100 |      100 |                   |
 zad2.js  |      100 |      100 |      100 |      100 |                   |
----------|----------|----------|----------|----------|-------------------|
➜ /workspace git:(master) X
```