Dežurni nastavnik: _____

Elektrotehnički fakultet u Beogradu
Katedra za računarsku tehniku i informatiku

Predmet: Testiranje softvera (SI3TS / 13S113TS)
Nastavnik: Prof. dr Dragan Bojić
Asistent: dipl. ing. Dražen Drašković
Ispitni rok: Januar 2018.
Datum: 19.01.2018.

Kandidat:* _____

Broj indeksa:* _____

Ispit traje 2.5 sata, a u toku prvog sata nije dozvoljeno napuštanje ispita.
Upotreba literature nije dozvoljena.

Zadatak 1	_____/4	*Zadatak 4*	_____/12
Zadatak 2	_____/10	*Zadatak 5*	_____/10
Zadatak 3	_____/12	*Zadatak 6*	_____/12

Ukupno na ispitu: **_____/60** ***Ukupno na domaćem*:*** **_____/40**

Rok u kome je odbranjen domaći*: **_____** (primer: januar 2018)

Ukupno: _____/100

Ocena: _____ (_____)

Napomena: Ukoliko u zadatku nešto nije dovoljno precizno definisano, student treba da uvede razumnu pretpostavku, da je uokviri (da bi se lakše prepoznala prilikom ocenjivanja) i da nastavi da izgrađuje preostali deo svog odgovora na temeljima uvedene pretpostavke. Kod pitanja koja imaju ponuđene odgovore treba **samo zaokružiti** jedan odgovor. Na ostala pitanja odgovarati **čitko, kratko i precizno**. * popunjava student.

---

1. [4] Nabrojati i kratko (na primeru) objasniti različite pristupe problemu predikcije rezultata testa.

2. [10] Za zadati metod *speedingfine*:

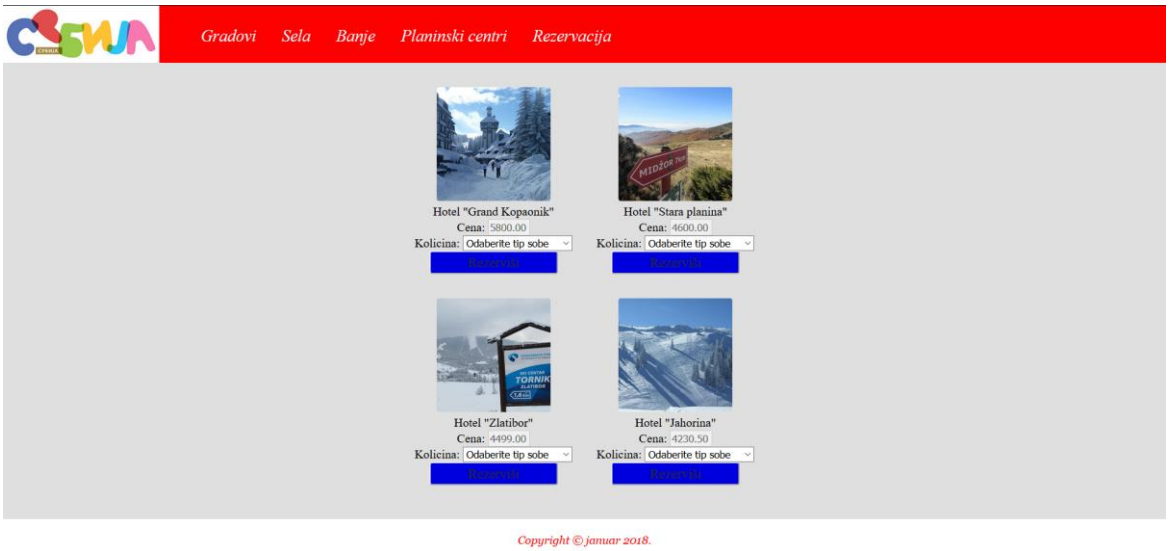
```
1 public static int speedingfine (int age, int overspeed;
  int licencemark) {
2     int fine = 0;
3     if ((age >= 25) && (overspeed < 30) && (licencemark < 3)) {
4           fine = fine + 100 * overspeed;
5     }
6     else {
7           if ((age < 25) || (licencemark >= 3)) {
8                 fine = fine + (200 * overspeed);
9           }
10          if (overspeed >= 30) {
11                fine = fine + 5000;
12          }
13    }
14    return fine;
15 }
```

a) Nacrtati graf kontrole toka i izračunati *McCabe*-ov broj.
b) Definisati test primere potrebne za postizanje 100% pokrivenosti linearno nezavisnih staza. Navesti eksplicitno koju putanju pokriva svaki test primer.
c) Napisati sve DU lance za promenljive *age* i *fine*.

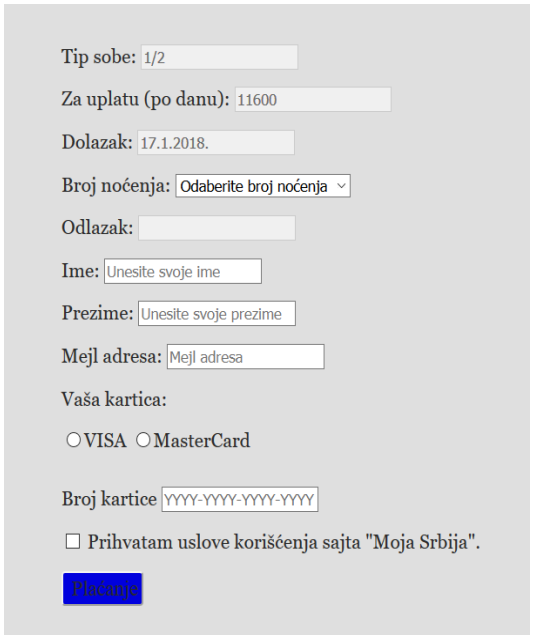
3. [12] Za metod *speedingfine* iz prethodnog zadatka (2):

    a) Navesti minimalan broj test primera za pokrivanje svih odluka. Za svaki test navesti (koristeći brojeve linija) programsku putanju koju test pokriva.
    b) Navesti minimalan broj test primera za pokrivanje svih prostih uslova. Za svaki test navesti (koristeći brojeve linija) pokriveni uslov i programsku putanju koju test pokriva.
    c) Navesti minimalan broj test primera za minimalno pokrivanje višestrukih uslova. Za svaki test navesti (koristeći brojeve linija) pokriveni uslov i programsku putanju koju test pokriva.

4. [12] Data je sledeća aplikacija za rezervaciju i plaćanje turističkih aranžmana putem portala "Moja Srbija".



Specifikacija zahteva: Svaka stavka menija u gornjem zaglavlju (*header*-u) vodi na zasebnu veb stranu. Unutar centralne sekcije svake veb strane nalaze se hoteli u kojima se može odsesti, kao što je prikazano na slici za veb stranu planinskih centara. Svaki hotel ima jednu ili više slika (ako ima više slike se menjaju nakon 5 sekundi), tekstualno polje sa cenom noćenja (po osobi), koja se povlači iz baze i ne može se ažurirati, i padajuću listu (svaka treba da ima 4 opcije: 1/2 dvokrevetna, 1/3 trokreventa, 1/4 četvorokrevetna, apartman, i inicijalnu opciju: Odaberite tip sobe). Potvrdom na dugme rezerviši, poziva se jedna funkcija (za svako dugme ista funkcija) koja izbacuje poruku: "Iznos vaše sobe po danu je: X RSD. Molimo Vas izvršite plaćanje odlaskom na veb stranu Rezervacija". X treba da predstavlja cenu_po_osobi x velicinu_sobe (npr. ako je vrednost po osobi 5800 i odabrana je dvokrevetna, iznos je 11 600 RSD). Ovaj iznos, kao i tip sobe, upisuje se u kolačić (*cookie*) i koristi na drugoj veb strani.



Kada korisnik ode na veb stranu rezervacija, u polju Dolazak treba automatski da se upiše današnji datum, a polje Odlazak treba da se generiše automatski, na osnovu izbora iz padajuće liste sa brojem noćenja (računa se kao: Datum_odlazak = Datum_dolazak + Broj_noćenja). Broj noćenja u padajućoj listi može biti: 1, 2, 3, 5, 7, 10 i 14. Polja tip sobe i za uplatu (po danu), treba da se iščitaju iz kolačića. Prilikom plaćanja smeštaja (kliknuti na dugme Plaćanje), treba izvršiti provere da li se korisnik saglasio sa uslovima korišćenja sajta (čekirao čekboks), da li je odabrao jedan tip kartice (pomoću radio dugmeta: Visa ili MasterCard) i da li postoji nešto upisano u polje za ime, za prezime i za adresu e-pošte (format mejl adrese mora biti: prezime@domen[.poddomeni].com ili ime.prezime@domen[.poddomeni].com). Kod kreditne kartice aplikacija izvršava sledeće provere: broj kartice mora imati 16 cifara, tako da između svake grupe od 4 cifre ide razmak ili srednja crta (-). Može se kombinovati, odnosno između nekih 4-cifrenih grupa da bude razmak, a između nekih crta. VISA mora počinjati prvom i završiti se poslednjom neparnom cifrom, a MasterCard mora počinjati sa brojem: 50, 51, 52, 53, 54 ili 55.

Moguće izlazne poruke su: (1) Niste saglasni sa uslovima korišćenja sajta "Moja Srbija". (2) Niste odabrali broj noćenja iz padajuće liste. (3) Molimo Vas izaberite Vašu karticu (u polju tip). (4) Broj bankovne kartice niste uneli u dobrom obliku! (5) Plaćanje se procesira. Iznos Vašeg aranžmana je: _____ RSD.

Strategijom testiranja crne kutije formirati klase ekvivalencije, kombinovano sa graničnim slučajevima, i napisati minimalan skup test primera prema tako formiranim klasama.

5. [10] Dati su delovi implementacija klasa za sortiranje: apstraktne klase *AbstractSort* i klase koja je iz nje izvedena, *QuickSort*:

```
abstract class AbstractSort{
	int v[];
	public AbstractSort(int nDim){
		v = new int[nDim];
		for (int i = 0; i<nDim; i++)
			v[i] = 0;
	}
	public void setVector(int nVector[]){
		for (int i=0; i<nVector.length; i++){
			v[i] = nVector[i];
		}
	}
	public void swap(int i, int j){
		int tmp;
		tmp = v[i];
		v[i] = v[j];
		v[j] = tmp;
	}
	abstract public void sort();
	...
}
```

```
class QuickSort extends AbstractSort
{
	public QuickSort(int nDim)
	{
		super(nDim);
	}

	private void quick(int low, int high) {
		int i = low, j = high;
		if(high<0)return;
		int pivot = v[low + (high-low)/2];
		while (i < j) {
			while (v[i] < pivot) {
				i++;
			}
			while (v[j] > pivot) {
				j--;
			}

			if (i <= j) {
				swap(i, j);
				i++;
				j--;
			}
		}
		if (low < j)
			quick(low, j);
		if (i < high)
			quick(i, high);
	}
```

```
        public void sort()
        {
              quick(0, v.length-1);
        }
}
```

a) [5] Napisati test primere koji uspešno testiraju petlje u metodi *quick(int,int)*. Za svaki test potrebno je definisati dužinu niza celih brojeva i sve elemente niza.

b) [5] Napisati JUnit testove koji bi testirali izvršavanje ove klase *QuickSort* i njenih metoda. Ukoliko je neophodno možete implementirati još neku dodatnu *getter* metodu u klasi.

6. [12] Dat je sledeći programski kod u programskom jeziku Java:

```
import java.util.Scanner;

public class MatrixMultiplication {
   public static void main(String[] args) {
       Scanner s = new Scanner(System.in);
       System.out.print("Enter number of rows in A: ");
       int rowsInA = s.nextInt();
       System.out.print("Enter number of columns in A / rows in B: ");
       int columnsInA = s.nextInt();
       System.out.print("Enter number of columns in B: ");
       int columnsInB = s.nextInt();
       int[][] a = new int[rowsInA][columnsInA];
       int[][] b = new int[columnsInA][columnsInB];
       System.out.println("Enter matrix A");
       for (int i = 0; i < a.length; i++) {
           for (int j = 0; j < a[0].length; j++) {
               a[i][j] = s.nextInt();
           }
       }
       System.out.println("Enter matrix B");
       for (int i = 0; i < b.length; i++) {
           for (int j = 0; j < b[0].length; j++) {
               b[i][j] = s.nextInt();
           }
       }
       int[][] c = multiply(a, b);
       System.out.println("Product of A and B is");
       for (int i = 0; i < c.length; i++) {
           for (int j = 0; j < c[0].length; j++) {
               System.out.print(c[i][j] + " ");
           }
           System.out.println();
       }
   }

   public static int[][] multiply(int[][] a, int[][] b) {
       int rowsInA = a.length;
       int columnsInA = a[0].length; // same as rows in B
       int columnsInB = b[0].length;
       int[][] c = new int[rowsInA][columnsInB];
       for (int i = 0; i < rowsInA; i++) {
           for (int j = 0; j < columnsInB; j++) {
               for (int k = 0; k < columnsInA; k++) {
                   c[i][j] = c[i][j] + a[i][k] * b[k][j];
               }
           }
       }
       return c;
   }
}
```

a) Da li ovaj programski kod može da se testira kao:

1) intra-metod testiranje _____

2) inter-metod testiranje _____

3) intra-klasno testiranje _____

4) inter-klasno testiranje _____

Napomena: *Zaokružiti broj ispred mogućeg testiranja i na crti pored napisati kao bi se testiralo.*

Da li bi se nešto promenilo (1-4) ukoliko bi postojala još jedna klasa *Matrica*, koja sadrži dvodimenzionalni niz celobrojnih elemenata? Odgovoriti sa DA/NE uz kratko obrazloženje.

b) Nacrtati graf(ove) koji reprezentuje gorenavedeni objekno-orijentisani kod.

c) Napisati test primere kojim biste postigli najveću pokrivenost (*coverage*) ovog programskog koda.

d) Da li je u programu neophodno staviti neki mehanizam hvatanja i obrade izuzetaka? Ukoliko je potrebno napisati u gornjem programskom kodu gde i koja su sva potencijalna mesta da se desi softverski defekat (bag) pri izvršavanju programa, kao i tip defekta?

e) Neka su u metodi *multiply(**int**[][] a, **int**[][] b)* sve FOR petlje zamenjene sa WHILE petljama, a zatim primenjen mutacioni operator SWDD, kojim se sve WHILE-DO petlje zamenjuju sa DO-WHILE. Da li bi primena operatora SWDD uticala na test primere iz tačke c) i kako? Objasniti i za takve primere napisati mutacioni skor, na osnovu dobijenih mutant programa.