1. Šta je jedinica, a šta jedinično testiranje softvera?

Jedinično testiranje je testiranje pojedinačnih objekata. Dakle, imamo test koji testira odgovarajuću klasu, odnosno metode koje nešto rade u okviru te klase.

Jedinica je klasa koju testiramo. Klasu uvek pišemo da bude proizvod za sebe, da je možemo iskoristiti u bilo kom drugom projektu I da je bez problema implementiramo I primenimo. Ova osobina OOP je bitna za jedinično testiranje.

1. Navedite prednosti i nedostatke jediničnog testiranja softvera.

Jedinični testovi značajno povećavaju kvalitet koda, zato što je svaka pojedinačna klasa iztestirana. Ako imamo testove za svaku pojedinačnu klasu lakše je čitanje koda, ako nešto ne razumemo u kodu, možemo da pogledamo testove i da vidimo šta taj deo koda radi. Ako imamo testove, onda možemo bez problema da radimo naknadne refactoring-e, izmene koda, a da se ne bojimo da ćemo narušiti neku funkciju na odnos koji postoji. Nakon izvršenog refactoring-a pustimo ponovo test i, ako je sve u redu, znači da nismo narušili prvobitnu funkciju na odnos programa.

Prednosti:

* 1. Veći kvalitet koda
  2. Lakše čitanje koda
  3. Moguće raditi naknadne refaktoringe
  4. Mogućnost pravljenja testova prihvaćanja
  5. Lako se uklapa u procedure izrade softvera
  6. Povećava pouzdanost celog sistema
  7. Smanjuje količinu grešaka u kodu

1. Šta je moguće proveriti Junit metodama?

Ove mane su obično vezane za kod koji je već ranije napravljen po neobjektno orijentisanim principima, pa je nemoguće implementirati na njega jedinično testiranje koje je specijalno optimizovano i prilagođeno za objektno orijentisano programiranje. Pored toga, dosta je teško primeniti jedinično testiranje na GUI klasi. GUI klasa je u interakciji sa korisniko pa je teško napisati JUnit testove koji bi izvršili takva testiranja.

Testiraju se samo klase, a ne i okruženje u kome se radi. Npr. preko JUnit testiranja se testira kod Javin, ali ne i SQL naredbe baze.

Mane:

* 1. Ne može da se primeni na GUI klase
  2. Testira se samo klase ne i okurženje u kom kod radi (baze podataka, mreža, ...)
  3. Kod mora da bude rađen po strogim Objektno Orijentisanim principima
  4. Sporiji razvojni ciklus
  5. do 3x više koda

1. Navedite osnovne klase koje se koriste u Junit testovima?
   1. **TestCase**, koji predstavlja pojedinačni slučaj testiranja (dakle, naša klasa nasleđuje klasu TestCase, i pravi pojedinačni slučaj testiranja)
   2. **TestSuite,** je skup klasa koje se testiraju ( imamo klasu koja nasleđuje TestSuite, i ona u sebi može da sadrži više TestCase-ova)
   3. **TestRunner(BaseTestRunner)**, je klasa koja obezbeđuje funkcionalnost, odnosno pokretanje testova,
   4. **TestResults**, je klasa u koju se smeštaju rezultati testa.
2. Koje metode se nasleđuju i implementiraju unutar TestCase klase?
   1. **setUp()** , koja nam služi da u nju postavimo osnovne postavke (da se konektujemo na bazu, ili da se povežemo na mrežu, ili da učitamo neki dokument sa diska)
   2. **testXXX()** , metode koje testiraju odgovarajuće metode iz konkretne klase (njih može da bude više, i svaka test metoda se odnosi na neku metodu u konkretnoj klasi)
   3. **tearDown()** , je metoda koja se pokreće nakon završetka testova (u nju možemo da zatvorimo vezu ka bazi, ili vezu ka drugom serveru, ili bilo šta što je potrebno uraditi nakon izvršenih testova).
3. Šta je TestSuite?

TestSuite, je skup klasa koje se testiraju ( imamo klasu koja nasleđuje TestSuite, i ona u sebi može da sadrži više TestCase-ova),

1. Čemu služe assertXXX() metode? Navedite najčešće korišćene

Assert metode služe da se proveri ispravnost neke vrednosti.

U slučaju **assertEquals()** metode proveravamo da li neki broj odgovara očekivanom broju, sa određenom tolerancijom. Metod **assertTrue()** je test pomoću koga proveravamo da li je vrednost tačna. Metod **assertFalse()** je suprotan od metoda AssertTrue().

Pored toga što možemo da proveravamo vrednosti prostih tipova, moramo da imamo mehanizam i za proveru objekata. To su metode **assertNull() i assertNotNull(),** da bi videli da li objekat postoji ili ne postoji. Onda imamo metode **assertSame() i assertNotSame(),** odnosno da li su dva objekta isti objekat ili ta dva objekta nisu isti objekat.

1. Koji je redosled izvršavanja JUnit testova? Da li redosled može da se podešava?

JUnit upotrebljava deterministički, ali nepredvidljiv redosled (MethodSorted.DEFAULT). Ako želite da definišete redosled tetsiranja vaše test klase, treba da upotrebite @FixMethodOrder i specificirajte raspoložive sortere sa MethodSorters:

@FixMethodOrder(MethodSorters.JVM):

Ovaj primer ostavlja redosled metoda onakav kakav je proizvep JVM. Ovaj redosled se menje od izvršenja do izvršenja, jer je nepredvidiljiv.

@FixMethodOrder(MethodSorters.NAME\_ASCENDING)

Ovaj iskaz sortira test metode prema imenu metoda, po alfabetskom redosledu.

1. Kako se testiraju izuzeci?

**new ArrayList<Object>().get(0);**

Ovaj kod bi trebalo da izbaci izuzetak **IndexOutOfBoundsException**. Oznaka **@Test** ima opcioni prametar **“expected”** koji kao vrednost ima podklase od Trowable. Ako želimo da proverimo da li ArrayList izbacuje tačan izuzetak, trebalo bi da napišemo:

**@Test(expected= IndexOutOfBoundsException.class)**

Prametar expected treba koristiti sa oprezom. Gornji test će proći ako bilo koj kod u metodu izbacuje IndexOutOfBoundsException. Kod dužih testova,preorućljivo je da se upotrebi ExpectedException pravilo, koje proizvodi, pored izuzetka, i određenu poruku

**@Rule**

**public ExpectedException thrown = ExpectedException.none();**

**@Test**

**public void shouldTestExceptionMessage() throws IndexOutOfBoundsException {**

**List<Object> list = new ArrayList<Object>();**

**thrown.expect(IndexOutOfBoundsException.class);**

**thrown.expectMessage("Index: 0, Size: 0");**

**list.get(0); // execution will never get past this line**

**}**

Testiranje metoda sa try-cach šemom se vrši na sledeći način:

**@Test**

**public void testExceptionMessage() {**

**try {**

**new ArrayList<Object>().get(0);**

**fail("Expected an IndexOutOfBoundsException to be thrown");**

**} catch (IndexOutOfBoundsException anIndexOutOfBoundsException) {**

**assertThat(anIndexOutOfBoundsException.getMessage(), is("Index: 0, Size: 0"));**

**}**

**}**

1. Po čemu se razlikuje assertThat() od ostalih assertXXX() metoda?
   1. Čitljivija je sintaksa
   2. Omogućava negaciju date kombinacije s: (not(s)), ili kombinaciju either(s).or(t)) se preslikavaju u kolekciju (each(s)) ili (afterFiveSeconds(s))
   3. Čitljive poruke o grešci.

**assertThat([value], [matcher statement]);**

1. Kako se koriste parametrizovani testovi?

@RunWith(Parameterized.class)

public class FibonacciTest {

@Parameters

public static Collection<Object[]> data() {

return Arrays.asList(new Object[][] {

{ 0, 0 }, { 1, 1 }, { 2, 1 }, { 3, 2 },

{ 4, 3 }, { 5, 5 },{ 6, 8 }

});

}

private int fInput;

private int fExpected;

public FibonacciTest(int input, int expected) {

fInput= input;

fExpected= expected;

}

@Test

public void test() {

assertEquals(fExpected, Fibonacci.compute(fInput));

}

}

1. Šta su i kako se koriste pravila u jediničnim testovima?

Pravila dozvoljavaju fleksibilnu promenu definicije ponašanja svakog test metoda I test klasi. Testeri mogu da koriste ranije definisana pravila ili da ih proširuju, ili da definišu nova pravila.

TemporaryFolder pravilo dozvoljava kreiranje foldera i datoteka koje se automastki brišu po završetku testiranja metoda. Moze se koristiti I pravilo ExpectedException umesto argumenta expected uz oznaku @Test.

1. Šta je lokalizacija i zašto se koristi u savremenom razvoju softvera?

Mnogi softverski proizvodi se danas koriste u više zemalja, a korisnici koriste različite jezzike komunikacije. Zbog toga, potrebno je id a se GUI (Graphic User Interface) tj. grafički korisnički interfejs kreira na više jezika. Java omugućava izradu GUI korišćenje lokalnog jezika uz minimalno podešavanje i minimalni dodatni naporq. To Java postiže na sledeći način:

Java koristi Unicode, 16-tno bitnu šemu kodiranja slovnih i drugih znakova, koja je postaveljenu od strane Unicode konzorcijuma. Primena Unicode šifriranja znatno olakšava manipulaciju tekstova (Java tipa string) pisanih korišćenje različitih jezika.

1. Šta je zadatak klase Locale? Šta sadrži ova klasa?

Klasa Locale predstavlja geografski, politički ili kulturni region u kome se koristi neki specifični jezik koristi. Svaka Swing klasa ima svojstvo locale koje nasleđuje od klase Component.

Klasa Locale sadrži unapred definisane mnoge lokalne konstante. Za srpski jezik, kod je sr, a za Srbiju – RS. Postoje u oba standarda, i skraćenice za tri slova. One su za Srbiju SRB, a za srpski jezik – srp. Locale.UK i Locale.ENGLISH

1. Kako se identifukuje kod jezika, a kako kod države?

Kod jezika je malim slovima, a države velikim slovima.

Za srpski jezik, kod je sr, a za Srbiju – RS. Postoje u oba standarda, i skraćenice sa tri slova. One su za Srbiju SRB, a za srpski jezik – srp.

Uz pomoć metoda **getAvailableLocals()** klase Locale, možete dobiti sve moguće oznake podržanih zemalja i jezika.

1. Objasnite formatiranje datuma i vremena primenom SimpleDateFormat.

Podklasa za formatiranje datuma i vremena SimpleDateFormat omogućava vam da izaberete format po izboru korisnika. Kada se sa konstruktorom formira objekat klase SimpleDateFormat koji konvertuje objekat Date u tekst (string) sa željenim formatom:

**public SimpleDateForma(String pattern);**

Parametar pattern je tekst koji čine karakeri sa specijalnim značenjem. Na primer y-znači godina (year), M znači mesec, d- znači dan u mesecu, G označava eru, h – sat, m – minut. S – sekunda, a z – vremenska zona. NA primer, da bi se definiso tekst ”Trenutno vreme je 1997.11.12.AD u 04:10:18 PST” potrebno je koristiti pattern yyyy.MM.dd G ‘u’ hh:mm:ss z”:

1. Na koji način je moguće podesiti format brojeva (u zavisnosti od lokalnih podešavanja)?

Brojevi se različito obeležavaju u različitim zemljama. To je prvenstveno povezano sa načinom označavanja decimalne zapete, kao i sa označavanjem hiljada. Na primer, broj koji se u Srbiji označava sa 5.000,55, obeležava sa su SAD sa 5,000.55, u Nemačkoj sa 5.000,55 (kao i u Srbiji), a u Francuskoj sa 5 000,55

Određivanje formata za prikazivanje brojeva vrši se u Javi korišćenjem klase NumberFormat, koja je apstraktna klasa koja obezbeđuje metode za formatiranje brojeva.

**NumberFormat numberFormat = NumberFormat.getInstance(Locale.FRANCE);**

**System.out.println(numberFormat.format(5000.555));**

U drugom iskazu broj je napisan sa decimalnom tačkom umesto sa zapetom, jer Se u Javi koristi tačke za označavanje decimalnih brojeva. Tek primenom isntrukcija za lokalizaciju, možemo prikazati broj na način koji se koristi na određenom lokalitetu, tj. u određenoj zemlji ili na određenom jeziiku.

Za prikazivanje brojeva koji predstavljaju novčane jedinice koristi se metod: NumberFormat.getCurrencyInstance() koji prikazuje novčane jedinice u lokalnom formatu.

1. Kao najveće unapređenje jezika Java, koje funkcionalnosti uvodi Java 8?

Lambda operand je jedan od najisčekivanijih jezičkih promena u celoj verziji Jave 8. Lamda nam omogućava da tretiramo kod kao podatke i da tretiramo funkcionalnosti kao metode.

U najprostijem objašnjenju lamda predstavlja listu parametara, simbol -> i telo.

Arrays.asList( "a", "b", "d" ).forEach( e -> System.out.println( e ) );

Osim pozivanja metoda lambda nam takođe omogućava i da vratimo vrednost a da pritom ne koristimo return.

Java 8 donosi nam novu vrstu referenci na metode koristeći Lamdu. Sledeći primer predstavlja klasu Car sa definicijama četiri vrste metode.

public static class Car {

public static Car create( final Supplier< Car > supplier ) {

return supplier.get();

}

public static void collide( final Car car ) {

System.out.println( "Collided " + car.toString() );

}

public void follow( final Car another ) {

System.out.println( "Following the " + another.toString() );

}

public void repair() {

System.out.println( "Repaired " + this.toString() );

}

}

Prvi tip metode referencira konstruktor sa sintaksom Class::new ili ukoliko se koriste Generici Class<T>::new. Molim obratite pažnju da konstruktor nema argumente.

**final Car car = Car.create( Car::new );**

**final List< Car > cars = Arrays.asList( car );**

Drugi tip omogućava pristup koristeći Class::statiča\_metoda .

**cars.forEach( Car::collide );**

Treći tip metoda omogućava pristup koristeći Class::method.

cars.forEach( Car::repair );

I na kraju četvrta metoda je metoda kojoj se pristupa preko instnace klase.

final Car police = Car.create( Car::new );

cars.forEach( police::follow );

1. Šta je funkcionalni interfejs?

Funkcionalni interfejs ima samo jednu metodu. Kao takav on se lako može konvertovati u lamda rečenicu. Java Runnable i Callable interfejsi su čist primer funkcionalnih interfejsa. Kako bi razlikovali obične interfejse i funkcionalne interfejse u Javi 8 je dodata nova anotacija @FunctionalInterface.

Ukoliko postoje više od jedne metode u funkcionalnom interfejsu on više neće raditi. Ono što treba znati jeste da statičke i default metode ne predstavljaju problem za funkcionalan interfejs.

1. Šta je default() metoda (kao novina u Javi 8)?

Java 8 donosi novost vezanu za deklarisanje interfejsa a ta novost jeste kreiranje default I static metoda unutar interfejsa. Default metode služe da dodamo nove metode u postojeće interfejse bez da menjamo binarnu kompitabilnost interfejsa sa starim verzijama interfejsa.

1. Navedite osnovne novine u Javi 9.

Moduli sistema Java platforme (Java Platform Modul System) predstavljaju novu vrstu komponenti koje se mogu da koriste pri programiranju u Javi. Modul u Javi predstavlja kolekciju samo-objašnjivih podataka i programa.

Java 9 obezbeđuje uprošćen format za prikazivanja oznake verzije novog izdanja. Koristi se string koji pomaže da se jasnije razdvoji glavna (major), manja (minor), sigurnosna (security) i popravljena (patch) izdanja Jave. String čine četiri broja odvojenih tačkama. Novi format stringa sa verzijom izdanja izgleda ovako:

**MAJOR.MINOR.SECURITY.PATCH**

Factory metod za kolekcije

**List<Point> list = List.of(new Point(1, 1), new Point(2, 2), new Point(3, 3), new Point(4, 4));**