OPRPP 2 - 2022./2023. - MI

===== TEORIJA =====

1. Što je bazen dretvi i koja je njegova namjena?

Bazen dretvi (engl. thread pool) je mehanizam u programiranju koji se koristi za efikasno upravljanje i izvršavanje dretvi. Bazen dretvi sastoji se od skupa prethodno stvorenih dretvi koje su spremne za izvršavanje. Kada se u aplikaciji pojavi zadatak koji zahtijeva dretvu, aplikacija traži slobodnu dretvu iz bazena i dodjeljuje joj taj zadatak. Nakon završetka zadatka, dretva se vraća u bazen, a aplikacija se može koristiti za izvršavanje drugih zadataka.

Namjena bazena dretvi je poboljšanje performansi aplikacije kroz smanjenje opterećenja stvaranja i uništavanja dretvi. Stvaranje dretvi može biti skupo i trošiti vrijeme i resurse, a bazen dretvi omogućuje ponovnu uporabu dretvi koje su već stvorene, smanjujući tako vrijeme izvršavanja aplikacije. Također, bazen dretvi pomaže u sprečavanju preopterećenja sustava stvaranjem ograničenja za broj dretvi koje se mogu izvršavati u isto vrijeme. To omogućuje da se aplikacija bolje skalira i da se izbjegnu problemi koji nastaju kada se stvara previše dretvi, što može dovesti do pada performansi sustava ili čak do rušenja aplikacije.

Napišite pseudokod za najjednostavniju implementaciju bazena dretvi.

```
inicijalizacija_bazena_dretvi(veličina):
	kreiraj novu listu dretvi
	za i = 1 do veličina:
		kreiraj novu dretvu
		dodaj dretvu na kraj liste dretvi

izvrši_zadatak_u_bazenu_dretvi(zadatak):
	ako lista dretvi nije prazna:
		izvadi prvu dretvu iz liste dretvi
		izvrši zadatak na dretvi
		dodaj dretvu na kraj liste dretvi

koristi_bazen_dretvi():
	inicijalizacija_bazena_dretvi(10) # ili bilo koja druga željena veličina bazena dretvi
	# neki kod koji dodaje zadatke u bazen dretvi
	# ...
	# izvršavanje zadatka u bazenu dretvi
	izvrši_zadatak_u_bazenu_dretvi(zadatak)
```

2. Koristimo paralelizaciju poslova uporabom bazena dretvi iz podsustava Executors.

a) Napišite programsku definiciju sučelja Runnable i Callable.

```
public interface Runnable {
	public void run();
}

public interface Callable<V> {
	public V call() throws Exception;
}
```

Čemu služe i u čemu se razlikuju.

Sučelja Runnable i Callable služe za definiranje zadataka koji se mogu izvršavati u dretvi ili bazenu dretvi. Glavna razlika između ova dva sučelja je u tome što Runnable ne vraća nikakvu vrijednost, dok Callable vraća vrijednost određenog tipa podataka.

By: CrazyFreak (& ChatGPT)

Kada se želi izvršiti zadatak u dretvi, stvara se nova instanca objekta koji implementira sučelje Runnable ili Callable. Nakon toga, objekt se predaje dretvi ili bazenu dretvi koji će ga izvršiti u pozadini.

Sučelje Runnable koristi se kada je potrebno pokrenuti neki posao u pozadini bez očekivanja povratne vrijednosti. Ovo se često koristi kada je zadatak jednostavan i ne zahtijeva nikakav specifičan rezultat, ili kada se rezultat obrađuje izvan pozadinske dretve. Primjeri takvih zadataka mogu uključivati stvaranje, čitanje ili pisanje datoteka, ispisivanje na konzolu, ili neku drugu vrstu jednostavnog računanja.

Sučelje Callable, s druge strane, koristi se kada je potrebno pokrenuti posao u pozadini i vratiti specifičnu vrijednost kao rezultat. Ovo se često koristi kada se izračunava neki kompleksniji izračun ili kada je potrebno dobiti neku vrijednost izvana, a izračunavanje je složeno i/ili sporo. Primjeri takvih zadataka mogu uključivati izračunavanje matematičkih funkcija, pretraživanje baza podataka, ili bilo koju vrstu računanja koje zahtijeva više vremena i/ili resursa.

Ukratko, Runnable se koristi kada je potrebno izvršiti jednostavan posao u pozadini, dok se Callable koristi kada je potrebno izvršiti složeniji posao i vratiti specifičnu vrijednost kao rezultat.

b) Implementirajte metodu "double clacNorm(double[] elements)" td. se izračuna norma (korijen iz sume kvadrata svih elemenata) tog vektora paralelizacijom poslova oslanjajući se na sučelje Callable i odabrani bazen dretvi iz podsustava Executors.

```
import java.util.concurrent.Callable;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
import java.util.concurrent.TimeUnit;

public class VectorNormCalculator {
    private static final int THREAD_POOL_SIZE = 4; // odabir veličine bazena dretvi

    public static double calcNorm(double[] elements) {
        int numThreads = Math.min(THREAD_POOL_SIZE, elements.length); // određivanje broja
dretvi

        ExecutorService executor = Executors.newFixedThreadPool(numThreads); // stvaranje
bazena dretvi

        int chunkSize = elements.length / numThreads; // određivanje veličine dijelova vektora

        double sum = 0.0;
        try {
            for (int i = 0; i < numThreads; i++) {
                int start = i * chunkSize; // početni indeks dijela vektora
                int end = (i == numThreads - 1) ? elements.length : (i + 1) * chunkSize; //
krajnji indeks dijela vektora

                Callable<Double> worker = new VectorNormWorker(elements, start, end); //
stvaranje zadatka za dretvu

                sum += executor.submit(worker).get(); // pokretanje zadatka i čekanje na
povratnu vrijednost
            }
        } catch (Exception e) {
            e.printStackTrace();
        } finally {
            executor.shutdown(); // zatvaranje bazena dretvi
        }
```

```
        return Math.sqrt(sum); // računanje korijena iz sume kvadrata
    }

    private static class VectorNormWorker implements Callable<Double> {
        private final double[] elements;
        private final int start;
        private final int end;

        public VectorNormWorker(double[] elements, int start, int end) {
            this.elements = elements;
            this.start = start;
            this.end = end;
        }

        @Override
        public Double call() throws Exception {
            double sum = 0.0;

            for (int i = start; i < end; i++) {
                sum += elements[i] * elements[i]; // računanje sume kvadrata elemenata
            }

            return sum; // povratna vrijednost je sume kvadrata
        }
    }
}
```

3. Razmatramo MVC (Model-View-Controller) paradigmu.

a) Objasnite kako izgleda obrada zahtjeva u MVC aplikaciji.

1. Korisnik stvara zahtjev putem korisničkog sučelja (View). Primjeri zahtjeva mogu uključivati klik mišem na gumb, upisivanje teksta u polje za unos ili odabir vrijednosti iz padajućeg izbornika.
2. View obavještava Controller o zahtjevu. To se obično radi putem nekog oblika slušača događaja (engl. event listener) koji će obrađivati događaje koje korisnik stvara.
3. Controller obradjuje zahtjev tako da dohvaća podatke iz Modela, obrađuje ih i/ili ažurira, te stvara odgovor na zahtjev. Ova faza obuhvaća poslovnu logiku aplikacije i glavnu obradu podataka.
4. Nakon obrade zahtjeva, Controller ažurira Model s potrebnim promjenama, ako ih ima.
5. View ažurira svoj prikaz na temelju ažuriranih podataka u Modelu. To uključuje osvježavanje prikaza na korisničkom sučelju kako bi korisnik vidio nove podatke.

b) Što treba nadodati u hw02 da bi se na adresi "/najveci?brojevi=10,30,8,25,17" dobili HTML dokument koji prikazuje najveći od poredanih brojeva.

// možda nije dobro

Novu klasu "NajveciServlet" koja nasljeđuje klasu HttpServlet i koja obrađuje HTTP zahtjeve za adresu "/najveci".

U metodi doGet() klase NajveciServlet treba izvući vrijednosti brojeva iz parametra "brojevi" u HTTP zahtjevu. To se može učiniti pozivanjem metode getParameter() nad objektom HttpServletRequest, koji se proslijeđuje kao parametar metode doGet().

Nakon dohvaćanja vrijednosti brojeva, treba ih pretvoriti u polje ili kolekciju brojeva i pronaći najveći broj. To se može učiniti pomoću standardnih Java metoda poput Integer.parseInt() i Collections.max().

Kada se pronađe najveći broj, treba ga prikazati u HTML formatu i poslati korisniku kao odgovor na HTTP zahtjev. To se može učiniti stvaranjem HTML koda u Java kodu, ili korištenjem nekog od Java templating rješenja (npr. Thymeleaf, JSP, itd.). HTML kod se zatim šalje korisniku putem odgovora HTTP zahtjeva.

4. Razmatramo web aplikaciju koja svakom korisniku na adresi "/app/koliko" prikazuje koliko je puta od pokretanja web preglednika posjetio tu stranicu.

Objasnite kako je ovo moguće izvesti?

Ovo je moguće izvesti korištenjem server-side tehnologija i zapisivanjem podataka u bazu podataka ili datoteke na poslužitelju.

Kada korisnik pristupi adresi "/app/koliko", aplikacija na serveru prima HTTP zahtjev s tom adresom.

Na poslužitelju se izvršava server-side kod (npr. servlet, PHP skripta ili Node.js aplikacija) koji prima taj zahtjev i provjerava da li postoji podatak o broju posjeta stranici.

Ako podatak ne postoji, server-side kod stvara novi podatak i postavlja ga na 1.

Ako podatak postoji, server-side kod povećava vrijednost tog podatka za 1.

Server-side kod zapisuje novu vrijednost podatka u bazu podataka ili datoteku na poslužitelju.

Server-side kod generira HTML stranicu s informacijama o broju posjeta i vraća ju korisniku kao odgovor na zahtjev.

Korisnik dobiva odgovor od servera i vidi prikaz broja posjeta na korisničkom sučelju.

Kako izgledaju HTTP zaglavlja prva dva puta (req-resp) kada korisnik zatraži navedenu stranicu ako prvi puta uopće pristupa toj web aplikaciji?

GET /app/koliko HTTP/1.1
Host: example.com
Connection: keep-alive
Cache-Control: max-age=0
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/89.0.4389.82 Safari/537.36
Accept:
text/html*,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9*
Accept-Encoding: gzip, deflate, br
Accept-Language: en-US,en;q=0.9

HTTP/1.1 200 OK
Date: Thu, 01 Apr 2021 12:00:00 GMT
Server: Apache/2.4.41 (Unix)
Content-Length: 27
Content-Type: text/html; charset=UTF-8

GET /app/koliko HTTP/1.1
Host: example.com
Connection: keep-alive
Cache-Control: max-age=0
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/89.0.4389.82 Safari/537.36
Accept:
text/html*,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9*
Accept-Encoding: gzip, deflate, br
Accept-Language: en-US,en;q=0.9

HTTP/1.1 200 OK
Date: Thu, 01 Apr 2021 12:01:00 GMT
Server: Apache/2.4.41 (Unix)
Content-Length: 27
Content-Type: text/html; charset=UTF-8

By: CrazyFreak (& ChatGPT)

===== ZADACI =====

1. Rješavate u hw02.

Jezik HTML omogućava da se stranici podesi pozadinska slika da se u zaglavlje dokumenta doda isječak oblika:

```
<head>
    <style>
        body { background-image: url('example.jpg'); }
    </style>
</head>
```

gdje URL pokazuje gdje se nalazi slika koju treba učitati i prikazati.
Uz pretpostavku da je Vaš poslužitelj pokrenut na portu 8000, osigurajte da se na adresi "http://localhost:8000/mi/vrijeme" ispisuje trenutno vrijeme.
Samu HTML stranicu MORA generirati smart skripta! Generirana stranica mora imati jednu od dvije moguće pozadinske slike (pripremite ih), te sadržavati link na novu stranicu na kojoj korisnik može odabrati koju od dvije pozadinske slike želi vidjeti na stranici s vremenom.

Važno 1: početno korisnik nije odabrao ništa pa se na stranici "vrijeme" svaki puta korisniku prikazuje nasumično odabrana pozadinska slika (koja se stoga svakim reloadom stranice potencijalno može mijenjati).
Važno 2: te dvije pozadinske slike unaprijed pripremite, i one se moraju dohvaćati s Vašeg poslužitelja. Korisnik ne može birati proizvoljnu sliku, već točno od ponuđene dvije.
Važno 3: ako poslužitelju pristupa više korisnika, svaki korisnik to mora moći obaviti za sebe. Na stranici za biranje korisniku treba prikazati obje slike i omogućiti mu da jedna na neki način odabere.

2. Rješavate u hw03.

Isti zadatak: (Jezik HTML omogućava...)

Razlika u odnosu na prvi zadatak je što se ovdje isti treba riješiti Javinim tehnologijama za izradu web-aplikacija, odnosno rezultat je valjana web-aplikacija.