# DOCUMENTAȚIE

TEMA 2

Nume : Rîpaș Bianca Ioana

Grupa: 30228

Cuprins

[1. Obiectivul temei 3](#_Toc95297885)

[2. Analiza problemei, modelare, scenarii, cazuri de utilizare 4](#_Toc95297886)

[3. Proiectare 6](#_Toc95297887)

[4. Implementare 10](#_Toc95297888)

[5. Rezultate 16](#_Toc95297889)

[6. Concluzii 18](#_Toc95297890)

[7. Bibliografie 19](#_Toc95297891)

1.Obiectivul temei

Obiectivul principal :

Proiectați și implementați o aplicație care are ca scop analiza sistemelor bazate pe cozi de așteptare prin (1) simulând o serie de N clienți care sosesc pentru service, care intră în Q cozi, așteaptă, sunt serviți și în final părăsesc cozile și (2) calculând timpul mediu de așteptare, timpul mediu de serviciu și ora de vârf.

Obiective secundare :

|  |  |
| --- | --- |
| Ttilu | Pagina |
| Analizați problema și identificați cerințele | 4 |
| Proiectați aplicația de simulare | 6 |
| Implementati aplicația de simulare | 10 |
| Testati aplicația de simulare | 16 |

2.Analiza problemei,modelare,scenarii,cazuri de utilizare

O diagramă USE-CASE este o reprezentare grafică a posibilelor interacțiuni ale utilizatorului cu un sistem.

Use case : simulare de configurare

Actor principal: utilizator

Principalul scenariu de succes:

1. Utilizatorul introduce valorile pentru: numărul de clienți,

numărul de cozi, intervalul de simulare, timpul minim și maxim de sosire și timpul minim și maxim de serviciu

2. Utilizatorul face clic pe butonul de incepere a simularii

3. Aplicația incepe simularea

Secvență alternativă: valori nevalide pentru parametrii de configurare - utilizatorul introduce valori nevalide pentru configurarea aplicației

- Utilizatorul apasa pe butonul Start , dar aceasta nu va functiona, se va arunca o exceptie in consola

- Scenariul revine la pasul 1

Utilizator. Interfata:

Setup

Start

Cozile

AVG timp

AVG asteptare

Cerințe funcționale:

- Aplicația de simulare ar trebui să permită utilizatorilor să configureze simularea

- Aplicația de simulare ar trebui să permită utilizatorilor să înceapă simularea

– Aplicația de simulare ar trebui să afișeze cozile în timp real evoluţie

– Aplicația de simulare ar trebui sa afiseze timpul de simulare , pentru a stii la ce timp ne aflam

– Aplicația de simulare ar trebui sa se opreasca cand ajunge la timpul de simulare

– Aplicația de simulare ar trebui sa afiseze in interfata grafica avg timp si avg asteptare

Cerințe nefuncționale:

- Aplicația de simulare ar trebui să fie intuitivă și ușor de utilizat de către utilizator

- Aplicația de simulare ar trebui să fie usor de inteles

- Aplicația de simulare ar trebui să fie usor de urmarit in timp real , sa fie o explicatie clara a evolutiei , o evolutie grafica ar fii de preferat

3.Proiectare

Structurile de date folosite sunt ArrayList<> si LinkedList<>.

Structura de date folosita pentru stocarea clientilor intr-o coada este „ LinkedList ” . Am ales aceasta structura deoarece putem folosi metodele add(<T> x) ( adauga un element de tipul <T> la coada ), removeFirst() ( elimina si returneaza primul element din coada ), isEmpty() si get(i) ( returneaza elementul de la indexul i ) , metode care ne ajuta la implementarea cozii .

Am folosit ,,ArrayList”, deoarece este un tablou simplu si avem acces direct si usor la informatii.

Diagrama de clase :

Graphical user interface, application, table

Description automatically generated

Diagrama de pachete:

Graphical user interface, text, application

Description automatically generatedGraphical user interface, table

Description automatically generated Table

Description automatically generated

În momentul rulării aplicației, pe ecran va apărea interfața grafică cu butonul de Start, precum si field-urile care trebuie completate de catre utilizator.Se poate observa ca interfata grafica este foarte usor de inteles.

Graphical user interface, text

Description automatically generated

Aici , se vor introduce datele de intrare : numarul de clienti,timpul care este alocat simularii, numarul de cozi si intervalele de servire si in care ajunge clientul.In chenarul alb din dreapta vor fi afisate pe tot parcursul simularii informatii despre cozi in timp real( care coada este ocupata si de ce client ) , dar ele se vor si observa dupa ce utilizatorul va da click pe butonul Start, care incepe simularea intr-o alta interfata.

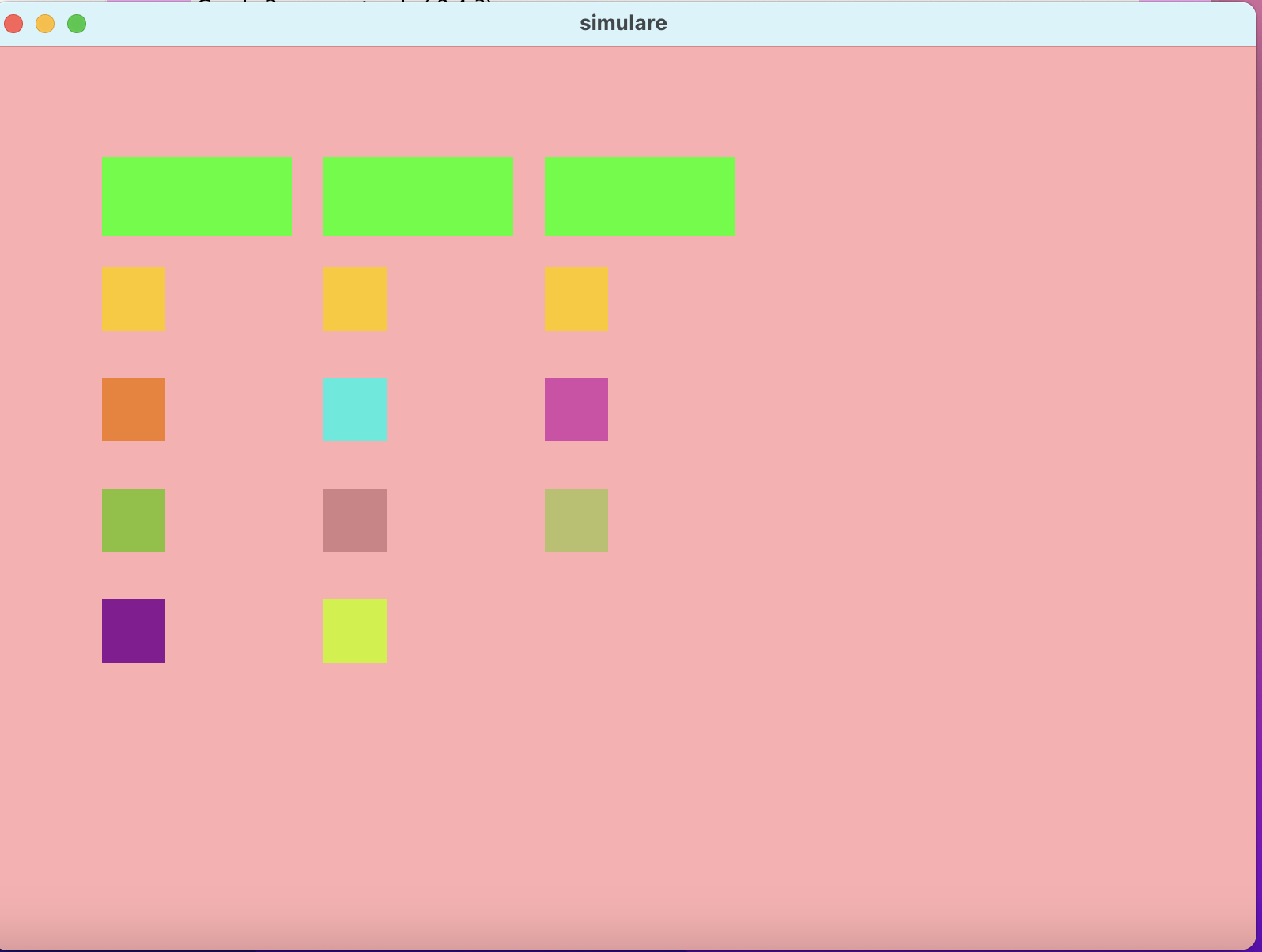
In casuta Timp va fii afisat timpul de la 0 pana la sfarsitul simularii, iar la sfarsit in ultimele doua casute vor fii afisate valoarea medie de asteptare si de servire.

Mai jos, este interfata a 2-a a programului.In momentul actual casele sunt rosii (nu sunt deschise), inseamna ca nu este niciun client.

Poza cu numarul 3 prezinta casele care sunt verzi , adica sunt deschise si au clienti.Primul client care este la casa este colorat cu portocaliu , iar ceilalti care sunt in asteptare sunt colorati in diferite culori Random .

A picture containing chart

Description automatically generated



4.Implementare

Proiectul meu se imparte in 3 pachete : Pachetul 1 date contine 3 clase : Client,Coada,Generator.

Pachetul 2 interfata contine : GUI si GUI1.

Pachetul 3 simulator contine Simulator.

Voi incepe cu clasa Client.

In aceasta clasa am declarat elementele private ce il caracterizeaza pe client , adica :

Graphical user interface

Description automatically generated

Dupa aceea, un constructor si get-ere pentru ID,arrivingTime si servingTime.

In aceasta clasa am implementat o singura metoda :

A picture containing text

Description automatically generated

getTimp(): returneaza suma dintre timpul de sosire si timpul de servire , adica timpul petrecut total de client la punctul de servicii.

Clasa Generator

Aceasta clasa este folosita pentru a genera obiecte de tip Client cu atributele specificate in clasa Client.Aceasta include un constructor care primeste valori pentru numarul de client,timp maxim si minim de servire, timp maxim si minim pentru timpul de sosire.De asemenea, clasa contine o metoda numita generare care primeste ca parametrii valorile maxime si minime pentru timpul de servire si sosire si genereaza un nou obiect de tip Client cu aceste valori , precum si un ID unic pentru fiecare client.Pentru a genera valorile aleatorii necesare , clasa utilizeaza clasa Random.

Graphical user interface, text, application, email

Description automatically generated

Clasa Coada

Clasa Coada reprezinta o coada ce poate fii utilizata pentru a gestiona sosirea si deservirea clientilor intr-un anumit punct de servicii.Aceasta contine o variabila de instanta private k , care este utilizata pentru a indica daca coada este deschisa sau nu , si o lista legata de clienti pe nume coada.

Metodele obisnuite sunt : getSize() , care returneaza numarul de client din coada,

getClient(int ID, int servingTime, int arrivingtTme) care returneaza un obiect de tip Client cu ID-ul,timpul de servire si sosire specificate,getCoada() returneaza lista de client din coada.

adaugaClient(Client client) adauga un client nou in coada , iar stergeClient(): elimina clientul din coada.

Text

Description automatically generated

Aici folosesc metodele specific structurii de date LinkedList removeFirst() si add ().

Metodele isOpen(): returneaza valoarea k , care indica daca coada este deschisa sau nu , iar open(boolean k) deschide coada si seteaza valorea variabilei k la true.

Graphical user interface, text, application

Description automatically generated

Cea mai importanta metoda a acestei clase este run(), care utilizeaza thread-ul pentru a permite simularii de deservire a clientilor din coada.Aceasta ruleaza intr-un ciclu,thread-ul verifica daca coada este goala sau nu si asteapta o secunda daca este , pentru a simula asteptarea clientilor.Daca coada nu este goala,thread-ul asteapta timpul de deservire al primului client din coada si apoi il elimina din coada.

Graphical user interface, text, application, email

Description automatically generated

Acum trecem la pachetul 2 care contine doua clase GUI si GUI1 care sunt responsabile pentru crearea interfetelor grafice.

Clasa GUI

Aceasta clasa implementeaza prima interfata grafica , care este prezentata utlizatorului. Aceasta contine campurile unde utilizatorul adauga datele de intrare.Despre felul in care arata am vorbit mai mult anterior , acum voi explica cum am facut sa arate in acel fel.

In prima parte am declarat etichetele pentru a stii utilizatorul in care JtextField sa introduca datele corespunzatoare fiecarei informatii .

**private** JLabel **l1** = **new** JLabel(**"Timp Simulare"**);

**private** JTextField **maxs** = **new** JTextField(**""**);

Aici am denumit frame-ul afisat utilizatorului.

**private** JFrame **frame** = **new** JFrame(**"Queue"**);

Aici am declarant chenarul alb din interfata in care scriu informatii in timp real.

**public static** JTextArea *write* = **new** JTextArea();

Declararea butonului :

**private** JButton **button** = **new** JButton(**"Start "**);

Aici am setat dimensiunile etichetei si JTextField-ului corespunzator ei , si le-am pus in interfata cu getContentPane.add().

**l1**.setBounds(24, 50, 126, 30);  
**timps**.setBounds(205, 50, 100, 30);  
**frame**.getContentPane().add(**l1**);  
**frame**.getContentPane().add(**timps**);

In codul de mai jos,am implementat functionalitatea butonului , adica atunci cand butonul Start este apasat se creeaza un obiect test de tip Simulator , care ia ca parametrii datele introduse de utilizator in JTextField-uri, cu ajutorul .getText().

**button**.addActionListener(**new** ActionListener() {  
 **public void** actionPerformed(ActionEvent e) {  
 Simulator test = **new** Simulator(Integer.*parseInt*(**nclienti**.getText()),Integer.*parseInt*(**ncozi**.getText()), Integer.*parseInt*(**mina**.getText()),  
 Integer.*parseInt*(**maxa**.getText()), Integer.*parseInt*(**mins**.getText()),  
 Integer.*parseInt*(**maxs**.getText()), Integer.*parseInt*(**timps**.getText()));  
 test.start();  
 }  
 });

Metoda main este metoda principala a clasei , care creeaza o instanta a clasei GUI si afiseaza fereastra grafica.

**public static void** main(String arg[]) {  
 GUI interfata = **new** GUI();  
 interfata.**frame**.setVisible(**true**);  
  
}

Clasa GUI1

Aceasta clasa extinde JPanel si are cateva metode pentru a desena grafic clientii si casele.Clasa este utilizata pentru afisarea cozilor din timpul simularii.

Metoda paintComponent() deseneaza elemente pe panou,ea primeste un obiect Graphics ca argument si il foloseste pentru a desena case si clienti.Variabilele x si y reprezinta coordonatele pentru desenarea elementelor.Utilizez un for pentru a itera prin cozi si clientii din acestea , iar daca o coada este goala , se desenaza o casa rosie , altfel se deseneaza o casa verde.Daca o coada are clienti, acestia sunt desenati ca patrate colorate aleatoriu sau portocaliu pentru primul client.

**public void** paintComponent(Graphics graphics) {  
 **super**.paintComponent(graphics);  
 **int** x = 70;  
 **int** i=0;  
 **int** j=0;  
 **for** ( i = 0; i < Simulator.*cozi*.**length**; i++) {  
 **int** y = 70;  
 **if**(Simulator.*cozi*[i].getSize()==0)  
 {  
 paintCasa(graphics, x, y);  
 }**else**{  
 paintCasa2(graphics,x,y);  
 }  
 **for** ( j = 0; j < Simulator.*cozi*[i].getSize(); j++) {  
 y += 70;  
 **if**(j==0)  
 {  
 paintClient2(graphics,x,y);  
 }**else** {  
 paintClient(graphics, x, y);  
 }}  
 x += 140;  
 }  
}

Metoda paintClient deseneaza un client cu o culoare aleatoare in patratul specificat.Metoda paintClient2 deseneaza primul client intr-coada cu culoarea portocalie.Metoda paintCasa deseneaza o casa goala cu culoarea rosie,iar metoda paintCasa2 deseneaza o casa cu client de culoare verde.Toace aceste patru metode folosesc obiectul Graphics pentru a desena formele pe panou.

**private void** paintClient(Graphics color1, **int** x, **int** y) {  
 Random color = **new** Random();  
 Color randomColor = **new** Color(color.nextFloat(), color.nextFloat(), color.nextFloat());  
 color1.setColor(randomColor);  
 color1.fillRect(x, y, 40, 40);  
 repaint();  
}

**private void** paintCasa(Graphics color, **int** x, **int** y) {  
 color.setColor(Color.***RED***);  
 color.fillRect(x, y, 120, 50);  
 repaint();  
}

Ultimul pachet al acestei aplicatii este simulator , care contine o singura clasa cu acelasi nume.

Simulator

Aceasta clasa este folosita pentru a simula interactiunea dintre client si cozile de asteptare, ea contine variabile de instanta si metode.

Clasa extinde clasa Thread , cee a ce inseamna ca functionalitatea de baza a clasei metoda start() va porni firul de executie.In momentul cand este apelata metoda start() pe obiectul Simulator , firul de executie va porni si va incepe sa execute codul din metoda run() intr-un mod concurrent cu restul aplicatiei.De aceea aplicatia poate sa ruleze in parallel,sa gestioneze multiple sarcini in acelasi timp.

**public class** Simulator **extends** Thread

c.start();

Variabilele sunt : nrCozi ( numarul de cozi de asteptare ) , timpulS ( timpul total de simulare), maxServingTime (timpul maxim de servire a unui client), minServingTime ( timpul minim de servire a unui client),maxArrivingTime ( timpul maxim de sosire a unui client),minArrivingTime(timpul minim de sosire a unui client),listaClienti(o lista de client generate aleatoriu), cozi ( un array de cozi de asteptare) , nrClienti ( numarul total de clienti), timp ( timpul curent in simulare ), k ( contorul utilizat pentru a scrie in fisier doar o singura data anumite informatii despre simulare), avgS ( timpul mediu de servire a clientilor), avgW( timpul mediu de asteptare a clientilor) si interfata un obiect de tip GUI1.

Clasa contine un constructor, metoda getTimp() statica.

Aceasta metoda primeste un obiect Client si il adauga intr-un ArrayList numit listaClienti, lista pe care o folosesc pentru a itera prin toti clientii si a-i adauga in cozi.

**public void** adaugaListaClienti(Client client) {  
 **listaClienti**.add(client);  
}

Metoda public int getCoadaPreferabila() calculeaza coada cu cel mai mic numar de client si returneaza indexul acesteia.Clientii sunt adaugati la cea mai scurta coada.

**public int** getCoadaPreferabila() {  
 **int** minIndex = -1;  
 **int** minSize = Integer.***MAX\_VALUE***;  
 **for** (**int** i = 0; i < *cozi*.**length**; i++) {  
 **if** (*cozi*[i].getSize() < minSize) {  
 minIndex = i;  
 minSize = *cozi*[i].getSize();  
 }  
 }  
 **return** minIndex;  
}

In aceasta metoda public void run() este scrisa toata logica, este apelata cand incepe simularea.Se creeaza o fereastra cu o interfata grafica si se genereaza clienti cu ajutorul clasei Generator. Apoi, se porneste fiecare coada. In timp ce timpul simularii trece , se itereaza prin lista de client si se efectueaza urmatoarele actiuni :

-daca timpul curent este egal cu timpul de sosire al clientului , se alege coada preferata pentru client cu ajutorul metodei getCoadaPreferabila(), iar clientul este adaugat in coada.De asemenea , scriu un mesaj corespunzator intr-un fisier text si actualizez interfata grafica.

-se coloreaza interfata grafica in functie de clientii care sunt in asteptare si de cei care sunt serviti

PrintWriter writer = **new** PrintWriter(**new** FileWriter(**"Test1.txt"**,**true**));

Aici am deschis fisierul text pentru a adauga informatii in el.Am scris in el folosind writer.println()

Dupa aceea , am afisat informatii cu privire la ce timp care client este in asteptare si la ce timp care client este la o anumita coada si afisez si care coada.Aceste informatii le afisez si in interfata grafica cu GUI.*write*.append()

Adaug clientul la coada buna prin

*cozi*[coadaBuna].adaugaClient(client);

Ultimele doua metode sunt cele care calculeaza media de asteptare si de servire.

**public double** setWaiting() {  
 **int** totalTimpAsteptare = 0;  
 **for** (Client client : **listaClienti**) {  
 totalTimpAsteptare += client.getTimp();  
 }  
 System.***out***.println(**"Numar clienti: "** + **listaClienti**.size());  
 System.***out***.println(**"Total timp asteptare: "** + totalTimpAsteptare);  
 **double** avgTimpAsteptare = (**double**) totalTimpAsteptare / **listaClienti**.size();  
 **return** avgTimpAsteptare;  
  
}

Parcurg lista de clienti si in variabila totalTimpAsteptare adaug timpul clientului care este sosire + servire , metoda scrisa in clasa Client despre care am scris mai sus si pentru o buna verificare afisez in consola numarul total de clienti si timpul total de asteptare pentru a verifica calculul.Acelasi lucru este valabil si pentru metoda setServing , diferit fiind doar ca adun la variabila totalTimpServire : client.getServingTime().

totalTimpServire += client.getServingTime();

5.Rezultate

Rezultatele sunt scrise intr-un fisier .txt dupa fiecare simulare, in afara de AVG wait si AVG serv ,care sunt afisate in interfata.

Pentru test 1 avem :

A picture containing graphical user interface

Description automatically generated

Pentru test 2 avem :

Graphical user interface

Description automatically generated with medium confidence

Pentru test 3 avem :

Graphical user interface

Description automatically generated with medium confidence

6.Concluzii

In concluzie, cu ajutorul acestui proiect mi-am dezvoltat abilitatile de a scrie cod in Java , de a implementa o aplicatie care sa functioneze si sa o realizez pentru a fii usor de folosit pentru oricine.Am invatat un lucru foarte important ,cel de a avea pachete aranjate pe functionalitatile fiecarei clase.

Pe parcursul dezvoltarii aplicatiei am fost nevoita sa fac studii despre functionarea firelor de executie ( Thread-urilor ) si atat modului lor de lucru individual cat si despre sincronizarea lor pentru lucrul in grup .

Pe langa acestea , mi-am dezvoltat cunostiintele de grafica am invatat moduri de afisare a unor informatii in timp real utilizatorilor .Mai mult , am reusit prin apasarea unui buton sa am o noua interfata , acest lucru mi-a placut foarte mult ca am reusit sa dobandesc abilitatea de a face acest lucru.

7.Bibliografie

<https://www.simplilearn.com/tutorials/java-tutorial/thread-in-java>

<https://www.geeksforgeeks.org/data-structures/linked-list/>

<https://stackoverflow.com/questions/4246351/creating-random-colour-in-java>

<http://www.java2s.com/Tutorial/Java/0261__2D-Graphics/Fill3DRectangle.htm>

<https://www.geeksforgeeks.org/java-util-random-nextint-java/>