Univerzitet u Kragujevcu

Fakultet inženjerskih nauka

Seminarski rad

Softverski inženjering

Tema:

Dinamička reprezentacija stabla intervalnog pretraživanja (R Search Tree)

Predmetni profesor:

Nenad Filipović

Student:

Kosta Erić 561/2015

Predmetni saradnik:

Tijana Šušteršič

SADRŽAJ

1. POSTAVKA ZADATKA....2
2. OPIS DELOVA PROGRAMA SA IZVORNIM KODOM....3
3. UML DIJAGRAMI....11
3.1 DIJAGRAM SLUČAJEVA KORIŠĆENJA (USE CASE DIAGRAM)....11
3.2 DIJAGRAM SEKVENCI (SEQUENCE DIAGRAM)....12
3.3 DIJAGRAM AKTIVNOSTI (ACTIVITY DIAGRAM)....13
3.4 DIJAGRAM STANJA (STATE MACHINE DIAGRAM)....14
3.5 DIJAGRAM OBJEKATA (OBJECT DIAGRAM)....15
3.6 DIJAGRAM KLASA (CLASS DIAGRAM)....16
4. LITERATURA....17

1. POSTAVKA ZADATKA

Potrebno je implementirati klasu za dinamičku reprezentaciju stabla intervalnog pretraživanja (u kodu *RST*). Instanca klase RSearchTree nastaje yadavanjem vektora (niza) brojeva nad kojim se gradi stablo (gde vrednost svakog čvora predstavlja zbir vrednosti njegove dece) i ima metode:

- *intervalNodes* – vraća minimalni broj čvorova kojma je obuhvaćen zadati interval
- *intervalSum* – metoda računa zbir elemenata podniza određenog interalom
- metode za odgovarajuće ispisivanje i uništavanje stabla

Klasa TreeNode (u kodu *Node*) nastaje zadavanjem indeksa krajeva intervala koji obuhvata i i brojem koji označava sumu intervala, i ima metode:

- dodavanja i izbacivanje levog i desnog podstabla
- *covers* – proverava da li čvor pokriva zadati interval
- ostale metode potrebne za uspešno izvršavanje zadatka

Obezbediti odgovarajuće ispisivanje čvora.

Za prijavljivanje i oporavak od grešaka koristiti se mehanizmom izuzetaka.

Napraviti korisnički interfejs za zadavanje stabla i upita tipa zbir broje na intervalu.

Projekat je napisan u programskom jeziku *Java 10* i na razvojnom ogruženju *Eclipse Photon IDE*.

2. OPIS DELOVA PROGRAMA SA IZVORNIM KODOM

Osnovna klasa celog zadatka je klasa ***Node*** (slika 1) u kojoj se nalaze svi potrebni atributi i metode za kreiranje pojedinačnog čvora.

```
package rst;
public class Node
{

    private int start,end,value;

    private Node leftChild;
    private Node rightChild;

    protected Node(int start, int end)
    {
        this.start = start;
        this.end = end;
    }
    protected Node(int value, int start, int end)
    {
        this.value = value;
        this.start = start;
        this.end = end;
    }
```

Slika 1

Promenljive leftChild i rightChild su promenljive tipa objekta koji predstavljaju pokazivače na levo i desno podstablo (dete) datog čvora.

Klasa sadrži dva konstruktora kako bi bilo omogućeno odvojeno kreiranje listova i samih čvorova stabla. Promenljive *start*, *end* i *value* predstavljaju početak i kraj intervala koji obuhvata dati čvor, kao i njegovu vrednost.

Klasa takođe sadrži i operacije za kreiranje i brisanje levog i desnog podstabla prikazane na slici 2,

```
//---------LEFT CHILD-----------------------//
public void insertLeftChild(int start, int end)
{
    this.leftChild = new Node(start, end);
}
public void insertLeftChild(int value, int start, int end)
{
    this.leftChild = new Node(value, start, end);
}
public void deleteLeftChild()
{
    this.leftChild = null;
}
public Node getLeftChild()
{
    return this.leftChild;
}


//------------RIGHT CHILD--------------------//
public void insertRightChild(int start, int end)
{
    this.rightChild = new Node(start, end);
}
public void insertRightChild(int value, int start, int end)
{
    this.rightChild = new Node(value, start, end);
}
public void deleteRightChild()
{
    this.rightChild = null;
}
public Node getRightChild()
{
    return this.rightChild;
}
```

Slika 2

operaciju *covers* (slika 3) koja proverava da li dati čvor pokriva odgovarajući interval (granice intervala se unose kao parametri metode)

```
public void covers(int left, int right)
{
    if(start <= left && end >= right)
    {
        System.out.println("Pokriva.");
    }
    else System.out.println("Ne pokriva.");
}
```

Slika 3

kao i pomoćne metode (slika 4) koje omogućavaju pristupanje i izmenu promenljivih klase.

```
//-------VALUE------------//
public void setValue(int value)
{
    this.value = value;
}
public int getValue()
{
    return this.value;
}

//----------START--------------//
public void setStart(int start)
{
    this.start = start;
}
public int getStart()
{
    return this.start;
}


//------------END----------------//
public void setEnd(int end)
{
    this.end = end;
}
public int getEnd()
{
    return this.end;
}
//------------------------------------//
```

Slika 4

Metoda *printInfo* (slika 5) je pomoćna metoda koja na standardni izlaz štampa sve informacije o datom čvoru stabla.

```
public void printInfo()
{
    System.out.println();
    System.out.println("Vrednost: " + this.value);
    System.out.println("Granice intervala: " + this.start + " --> " + this.end);
    System.out.println("Vrednost levog deteta: " + this.leftChild.value);
    System.out.println("Vrednost desnog deteta: " + this.rightChild.value);
}
```

Slika 5

Klasa ***RST*** (slika 6) je klasa u kojoj se nalaze sve metode za kreiranje i izvrsavanje operacija nad stablom.

```
package rst;

import java.util.ArrayList;

public class RST {

    private int[] a;
    private int suma;
    Node root;
    static int count = 0;
    static int left,prleft;
    static ArrayList<Node> lista = new ArrayList<Node>();

    protected RST(int a[])
    {
        this.a = a;
        this.root = new Node(0, a.length - 1);
    }
```

Slika 6

Klasa sadrži konstruktor koji prima listu ulaznih cvorova (listova) nad kojima se gradi stablo, objekat tipa *Node* koji predstavlja koren celokupnog stabla kao i pomoćne promenljive koje se koriste u metodama klase.

Na slici 7 se nalazi implementacija metode *buildRST* koja kreira stablo. Metoda rekurzivno gradi stablo, od listova ka korenu. Dva osnovna slucaja kada rekurzija staje su slucaj kada su oba deteta datog cvoa listovi (kada je dužina intervala jednaka 1) ili kada dati cvog sadrži levo podstablo, a na mestu desnog podstabla se nalazi list (kada je dužina intervala jednaka 2). Prilikom kreiranja čvorova, njihove vrednosti se uporedo ažuriraju po zadatom pravilu.

```
public Node buildRST(Node n)
{
    int start = n.getStart();
    int end = n.getEnd();
    int mid = (end + start) / 2;

    if ((end - start) == 1)
    {
        n.insertLeftChild(a[start], start, start);
        count++;
        n.insertRightChild(a[end], end, end);
        count++;
        n.setValue(n.getLeftChild().getValue() + n.getRightChild().getValue());

        return n;
    }
    if ((end - start) == 2)
    {
        n.insertLeftChild(start, mid);
        count++;
        n.insertRightChild(a[end], end, end);
        count++;
        Node n1 = buildRST(n.getLeftChild());
        n.setValue(n.getRightChild().getValue() + n1.getValue());

        return n;
    }
    n.insertLeftChild(start, mid);
    count++;
    n.insertRightChild(mid + 1, end);
    count++;

    Node nl = buildRST(n.getLeftChild());
    Node nr = buildRST(n.getRightChild());

    n.setValue(nl.getValue() + nr.getValue());

    return n;
```

Slika 7

Metoda *destroy* (slika 8) rekurzivno unistava sve čvorove stabla, od listova ka korenu slika.

```
public void destroy(Node root)
{
    if(root != null)
    {
        destroy(root.getLeftChild());
        destroy(root.getRightChild());
        root.deleteLeftChild();
        root.deleteRightChild();
    }
    if(root == this.root)
    {
        this.root = null;
    }
}
```

Slika 8

Metoda *display* (slika 9) je pomoćna metoda koja izlistava sve elemente stabla (sve čvorove) prema *inorder* načinu prolaska kroz stablo, gde se prvo prikazuje levo dete, koren pa desno dete.

```
public void display(Node root)
{
    if (root != null)
    {
        display(root.getLeftChild());
        System.out.print(root.getValue() + " ");
        display(root.getRightChild());
    }
}
public void displayList()
{
    for(Node l:lista)
    {
        System.out.print(l.getValue() + " ");
    }
}
```

Slika 9

Metoda *displayList* (slika 10) je pomoćna metoda koja prikazuje sadržaj liste koju koriste metode *intervalNodes* i *intervalSum*.

```
public Node intervalNodes(int left, int right, Node n)
{
    if(left > right)
    {
        return n;
    }
    if(n.getStart() == left && n.getEnd() == right)
    {
        lista.add(n);
        RST.prleft = RST.left;
        return n;
    }else if(collide(n.getStart(), n.getEnd(), left, right) == -1)
    {
        lista.add(n);
        RST.left = n.getEnd() + 1;
        return n;
    }
    if(n.getLeftChild() != null)
```

Slika 10

Metoda *intervalNodes* (slika 10) kao ulazne argumente prima donju i gornju granicu intervala, za koji treba da se nađe minimalni broj čvorova stabla koji pokrivaju dati interval. Ova metoda koristi pomoćnu metodu *collide* (slika 11),

```
public int collide(int start, int end, int left, int right)
{
    if(start == left && end < right)
    {
        return -1;//left collide
    }else return 0;
}
```

Slika 11

koja proverava da li se određeni čvor sa leve strane graniči sa zadatim intervalom, a zatim se rekurzivno pretražuju elementi koji čine dati interval i čvorovi koji ga čine se ubacuju u listu. Prilikom ubacivanja čvora u listu donja granica traženog intervala se pomera na granicu koja je za jedan veća od gornje granice intervala koji pokriva dati čvor i time se nalaze čvorovi čiji međusobi zbir intervala daje početni traženi interval. Za ažuriranje granica intervala i čuvanje čvorova koji su potrebni koriste ste statičke promeljive jer je neophodno da postoji jedna kopija ovih promenljivih kako bi svi kreirani objekti zajednički koristili te promenljive (*left*, *prleft*, *lista*) (slika 12) .

```
if(n.getLeftChild() != null)
{
    if(left > n.getLeftChild().getEnd())
    {
        intervalNodes(left, right, n.getRightChild());
    }else if(left == n.getLeftChild().getEnd())
    {
        if(n.getLeftChild().getStart() == n.getLeftChild().getEnd())
        {
            lista.add(n.getLeftChild());
            RST.prleft = RST.left;
        }else if(n.getLeftChild().getRightChild().getLeftChild() == null)
        {
            lista.add(n.getLeftChild().getRightChild());
            left++;
            intervalNodes(left, right, n);
        }else
        {
            RST.left = left;
            RST.prleft = left;
            intervalNodes(left, right, n.getLeftChild());
            if(RST.prleft != RST.left) intervalNodes(RST.left, right, n);
        }
    }else if(left < n.getLeftChild().getEnd())
    {
        RST.left = left;
        RST.prleft = left;
        intervalNodes(left, right, n.getLeftChild());
        if(RST.prleft != RST.left) intervalNodes(RST.left, right, n);
    }
}else lista.add(n);

return n;
```

Slika 12

Metoda *intervalSum* (slika 13) se nadovezuje na prethodno opisanu metodu, i vraća zbir vrednosti elemenata koji pokrivaju zadati interval.

```
public int intervalSum(int left, int right, Node n)
{
    lista.clear();
    intervalNodes(left, right, n);
    for(Node k:lista)
    {
        suma += k.getValue();
    }
    lista.clear();
    return suma;
}
```

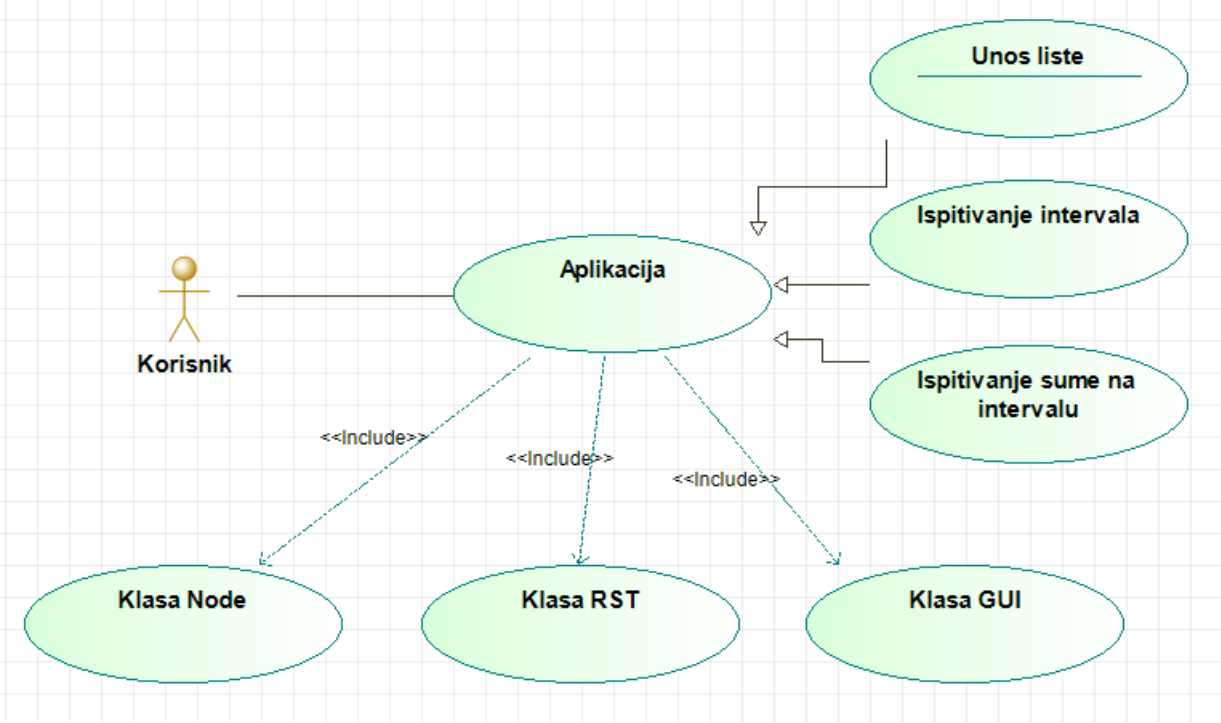
Slika 13

3. UML DIJAGRAMI

3.1 DIJAGRAM SLUČAJEVA KORIŠĆENJA (USE CASE DIAGRAM)

- Dijagram slučajeva korišćenja prikazuje skup slučajeva korišćenja i aktera.
- Vizuelizuje ponašanje sistema, podstistema ili interfejsa.

Dijagram slučajeva korišćenja za projektni zadatak je prikazan na slici 14.

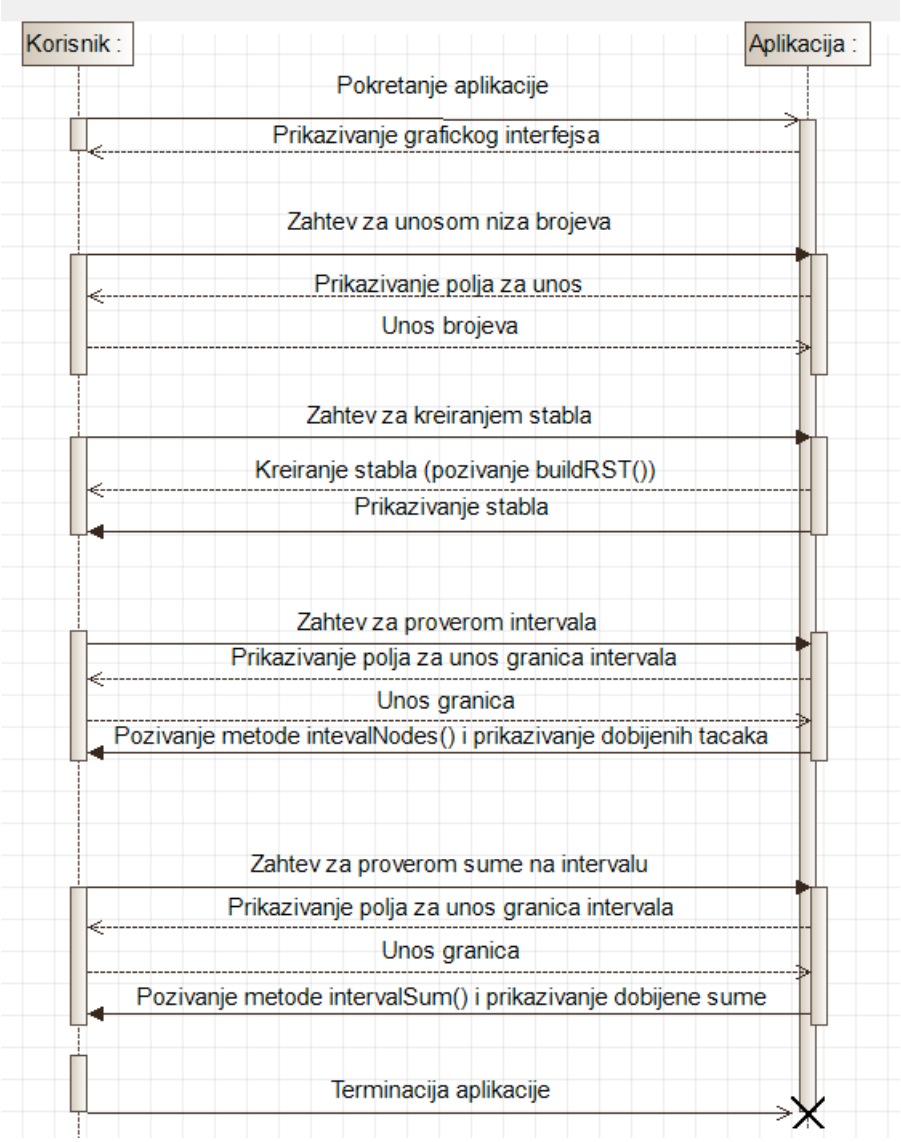


Slika 14

3.2 DIJAGRAM SEKVENCI (SEQUENCE DIAGRAM)

- Prikazuje komunikaciju između skupa objekata, koja se ostvaruje porukama koje objekti međusobno razmenjuju u cilju ostvarivanja očekivanog ponašanja.
- Detaljno opisuje kako se operacije izvode – koje poruke se šalju i kada
- Koristi se za prikaz jednog ili više scenarija

Dijagram sekvenci za projektni zadatak je prikazan na slici 15.

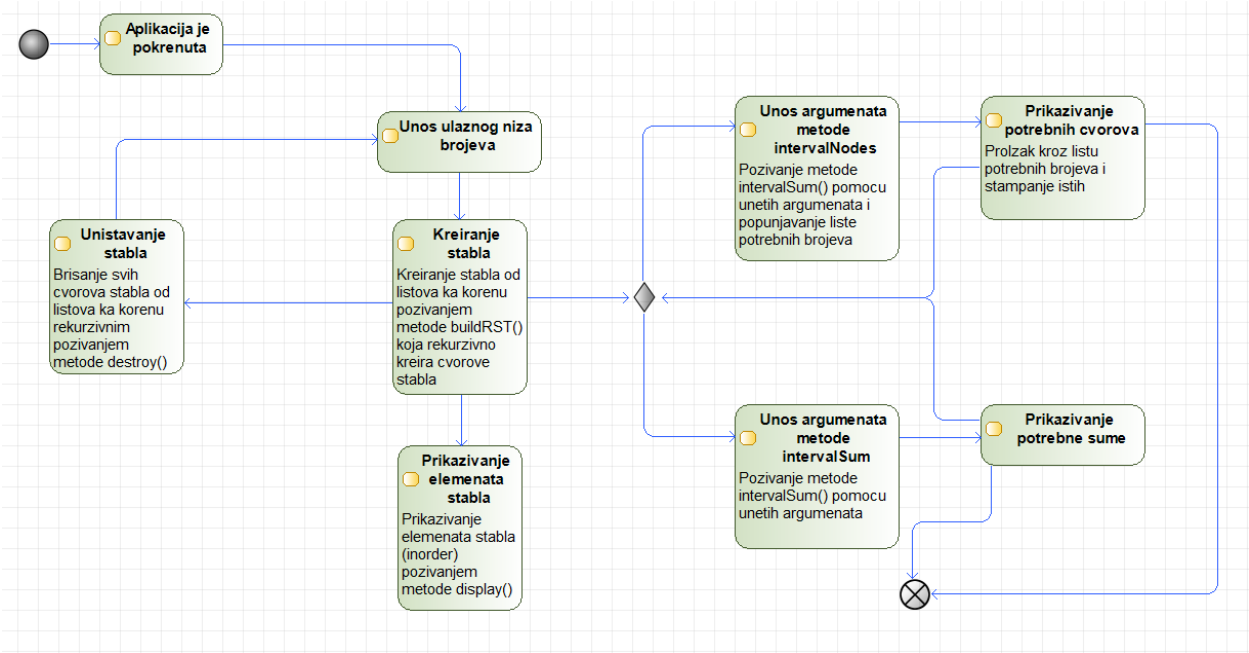


Slika 15

3.3 DIJAGRAM AKTIVNOSTI (ACTIVITY DIAGRAM)

- Dijagrami aktivnosti su namenjeni modeliranju dinamičkih aspekata (ponašanja) sistema
- Dijagram aktivnosti prikazuje:
  - ❖ Tok aktivnosti koju izvršavaju objekti
  - ❖ Eventualno i tok objekata između koraka aktivnosti

Dijagram aktivnosti za projektni zadatak je prikazan na slici 16.



Slika 16

3.4 DIJAGRAM STANJA (STATE MACHINE DIAGRAM)

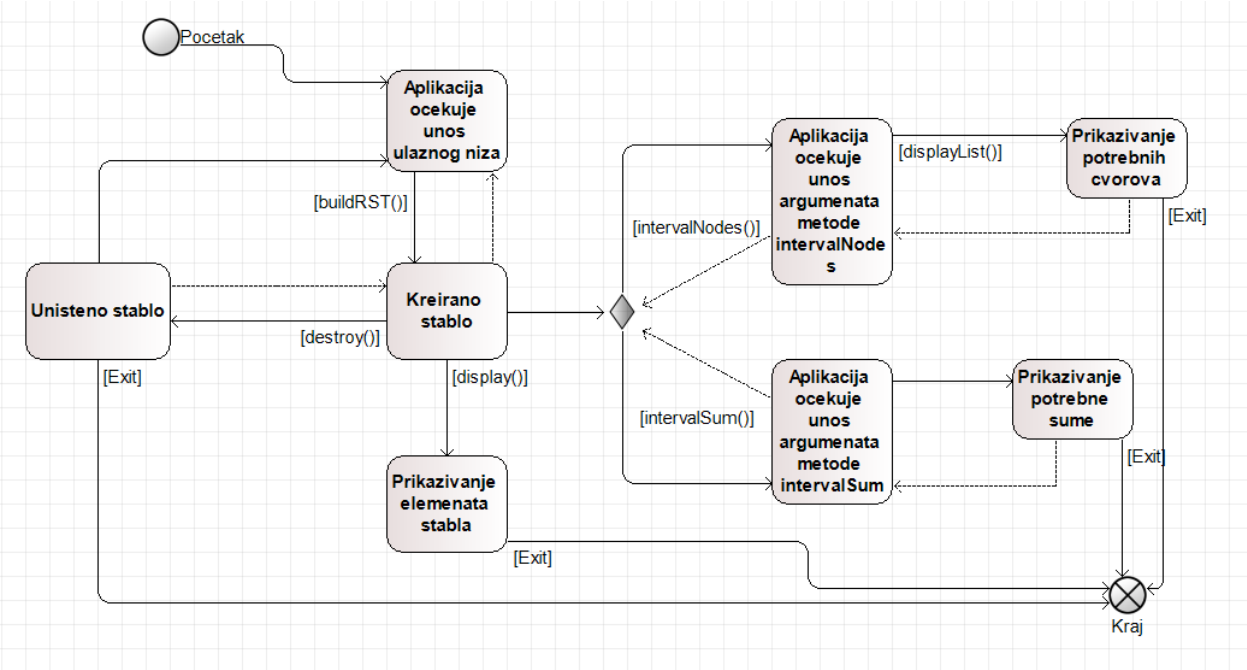
Automat stanja/ Mašina stanja (state machine)

- Ponašanje koje specificira sekvence stanja kroz koja prolazi
- Modelira ponašanje nekog entiteta ili protokol interakcije

Dijagram stanja je graf koji prikazuje automat stanja

- Čvorovi su stanja
- Grane su prelazi

Dijagram stanja za projektni zadatak je prikazan na slici 17.

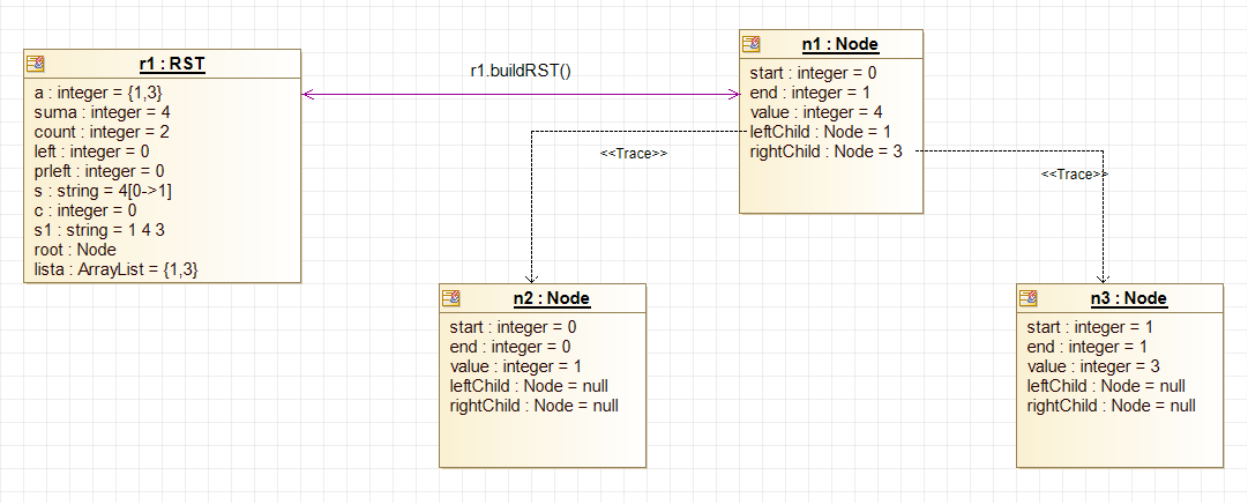


Slika 17

3.5 DIJAGRAM OBJEKATA (OBJECT DIAGRAM)

- Dijagrami objekata prikazuju primerke (objekte) apstrakcija (klasa) i njihove veze preko kojih objekti mogu da komuniciraju.
- Predstavljaju "snimak" pogleda na sistem u jednom trenutku (prikazuju se objekti sa trenutnim stanjem i trenutne veze)
- Element dijagrama objekata su:
  - ❖ Stvari (objekti i paketi)
  - ❖ Relacije: veze

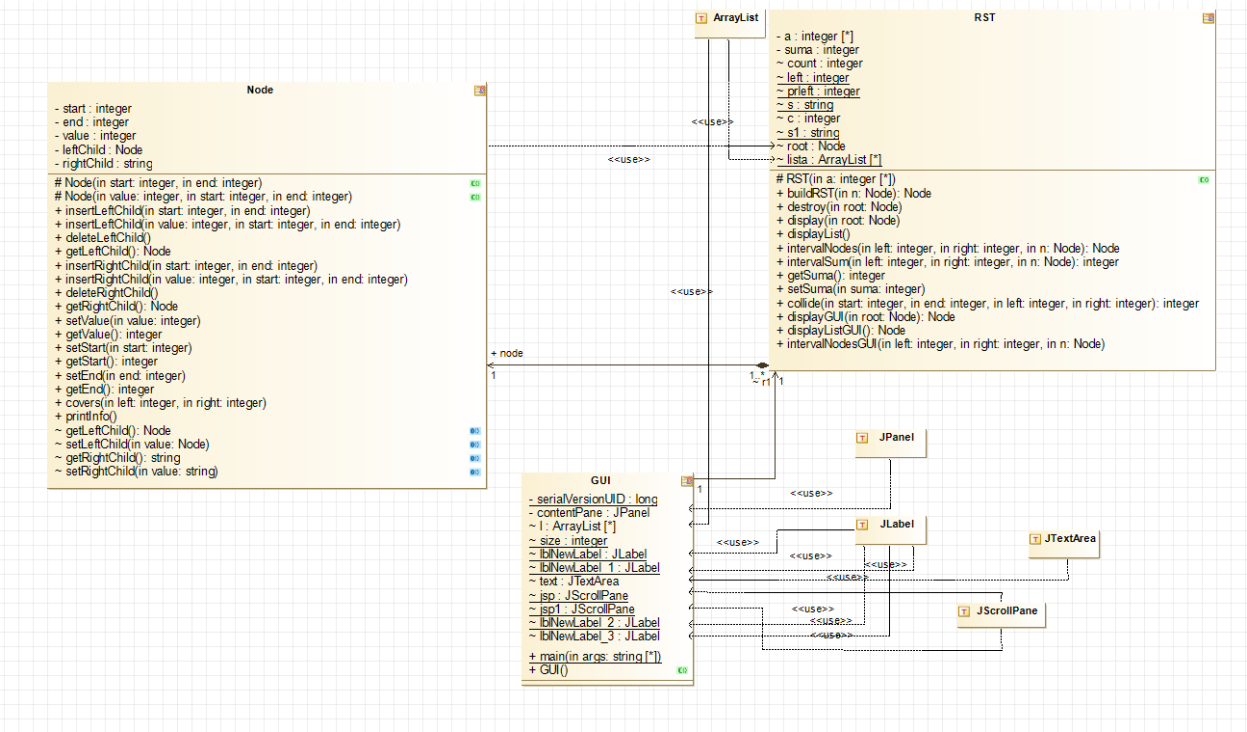
Dijagram objekata za projektni zadatak je prikazan na slici 18.



Slika 18

3.6 DIJAGRAM KLASA (CLASS DIAGRAM)

Dijagram klasa za projektni zadatak je prikazan na slici 19.



Slika 19

4. LITERATURA

- http://moodle.mfkg.rs/course/view.php?id=524
- https://docs.oracle.com/javase/10/docs/api/overview-summary.html
- https://www.eclipse.org/documentation/
- https://en.wikipedia.org/wiki/Interval_tree
- https://www.modelio.org/documentation-menu/user-manuals.html