

**KAUNO TECHNOLOGIJOS UNIVERSITETAS**

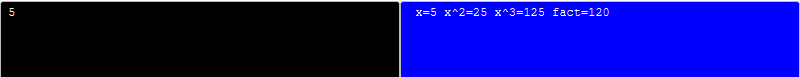
**INFORMATIKOS FAKULTETAS**

**DUOMENŲ STRUKTŪROS**

**1 LABORATORINIS DARBAS**

**Lab1\_A**

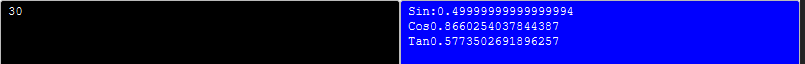
Dem0o\_Hello: 1



Blogai skačiuoja jeigu skaičius yra mažesnis nei 0.

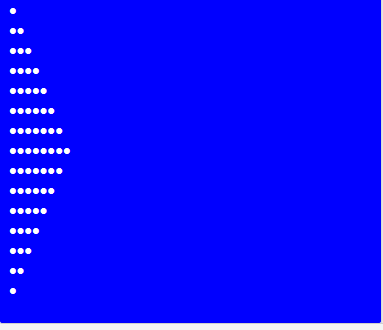
DemOo\_Hello: 2





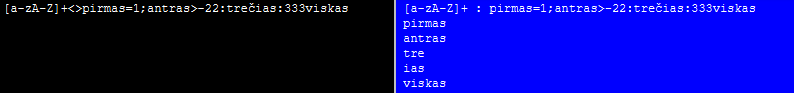
Demo1\_StringBuilder:1





Demo3\_Text:1

Atpažinti regex.  
Lotyniški žodžiai:



Lietuviškį žodžiai:

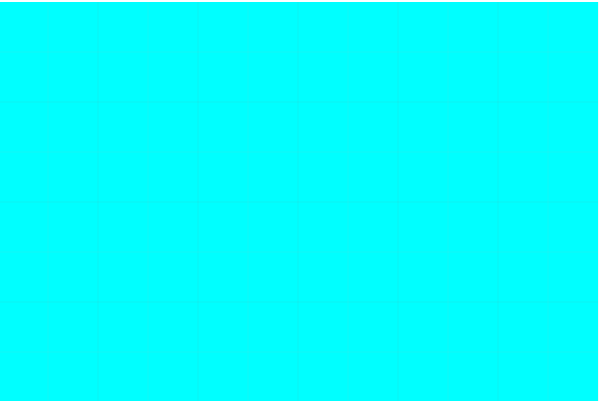


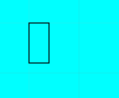
Informacija tarp skliaustu:



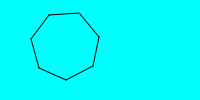
Demo0\_Basic:

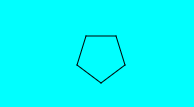
UŽDUOTIS\_1: plonomis linijomis su žingsniu step=50 nubrėžkite tinklelį



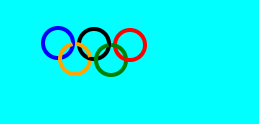


// UŽDUOTIS\_3: nubrėžkite taisyklingus 3, 4, 5, ..., 9-kampius

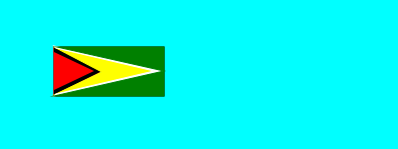
 7 kampu

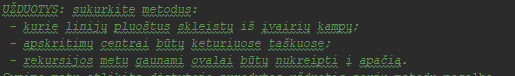
 5 kampu

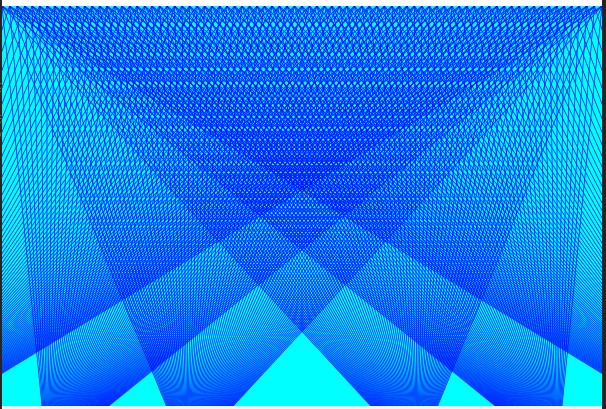


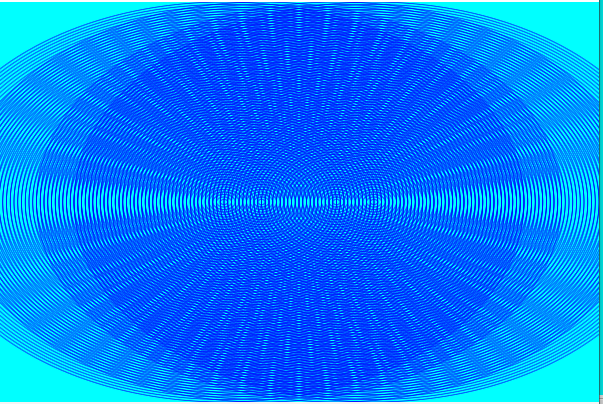
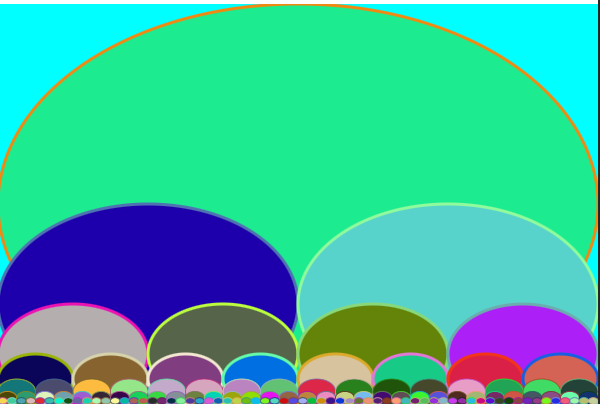




Guanos vėliava  
Demo1\_Raster

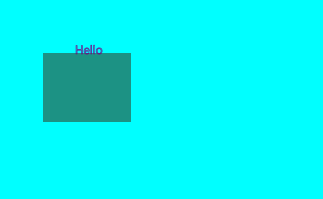


Pluoštų skleidimas iš įvairiu kampu

Apskritimu centrai keturiuose taškuose

Rekursijos metu ovalai nukreipti į apačia

// rinkinys figūrų, kurios brėžiamos atlikus transformacijas  
// UŽDUOTIS: sukurkite savo figūrų rinkinį, kurį transformuosite

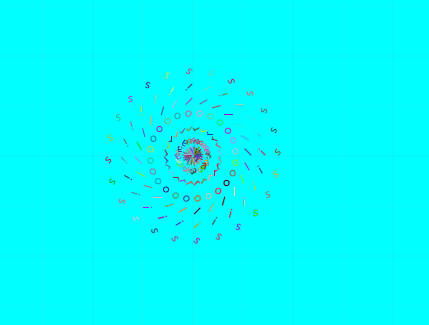




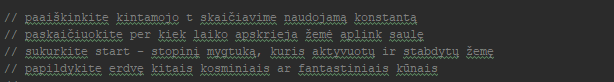


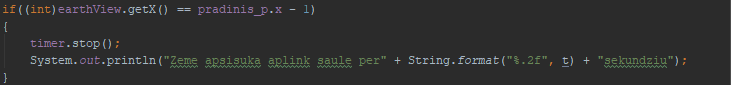


Mano rinkinys:

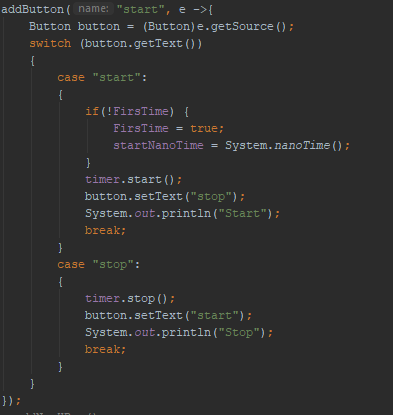


Demo6\_Images

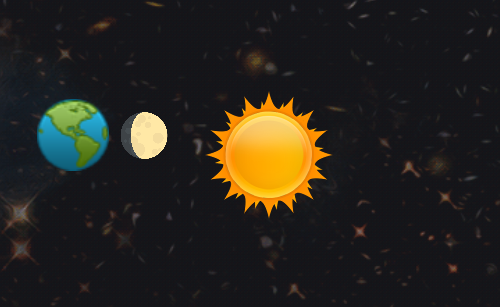
  
t – nurodo pradini laika, kada pradedame animacija vykdyti.

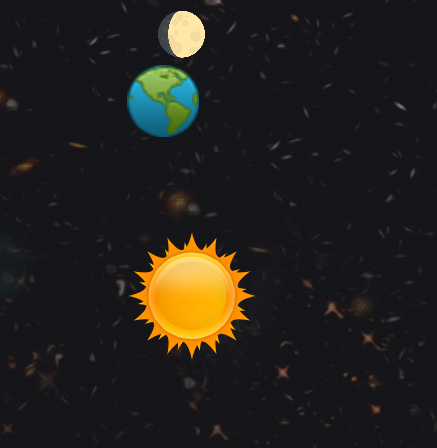






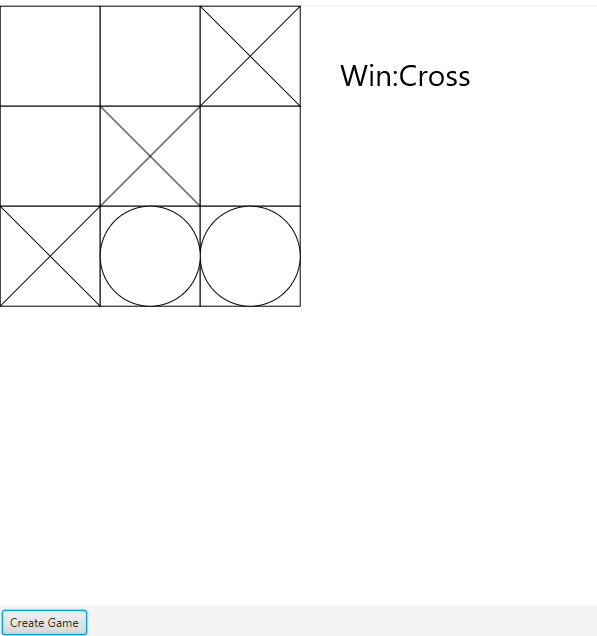
Dangaus kūnas - Mėnulis





Papildomas darbas:

Tic-Tac žaidimas



**Lab1\_B**

**Individualios užduotys:**

LinkedList metodai:

retainAll(LinkedList<E> c)

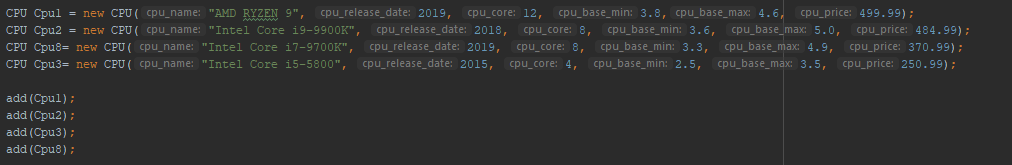
removeFirst()

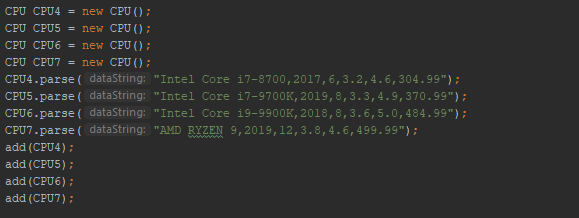
removeFirstOccurence(Object o)

**Greitaveika:**

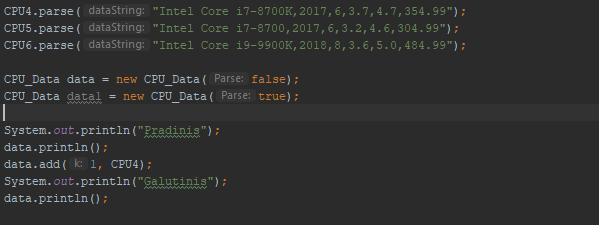
Math.pow(x, 0.5) <-> Math.sqrt(x)

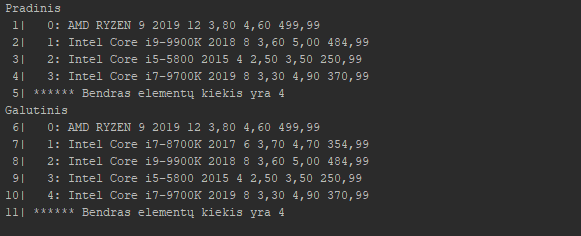
ArrayList<Integer> <-> LinkedList<Integer> metodas lastIndexOf(Object o)

Data – Duomenys 

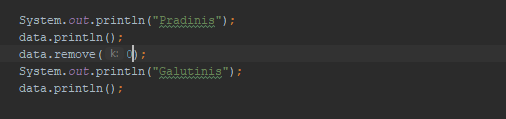
Data1 – Duomenys

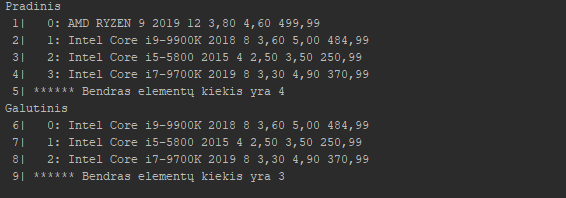
@Override  
public boolean add(int k, E e) {  
 if (e == null) {  
 return false;  
 }  
 if (k < 0 || k >= size) {  
 return false;  
 }  
 Node<E> node = new Node<>(e, null);  
 Node<E> currentnode = first;  
 if(k == 0)  
 {  
 node.next = currentnode.next;  
 currentnode.next = node;  
 return true;  
 }  
 for(int i =0; i < k; i++)  
 {  
 if(i + 1 == k)  
 {  
 node.next = currentnode.next;  
 currentnode.next = node;  
 }  
 currentnode = currentnode.next;  
 }  
 return false;  
}



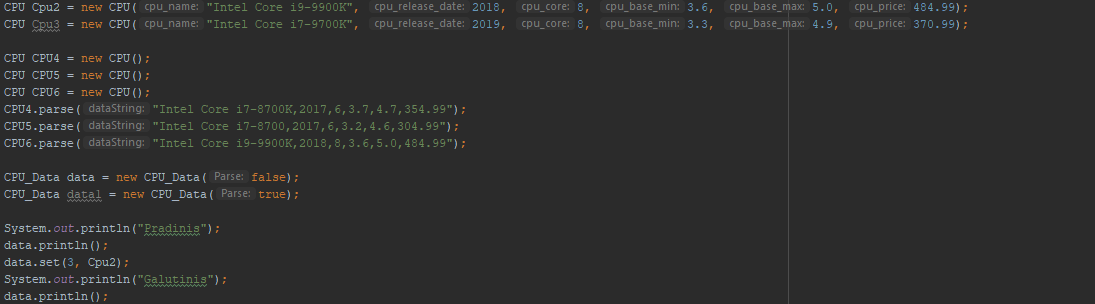


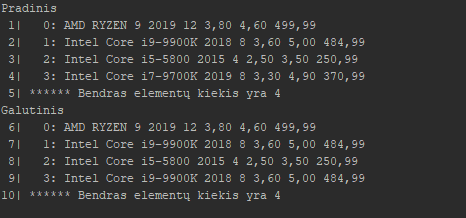
@Override  
public E remove(int k) {  
 if(k < 0 || k >= size) return null;  
 if(size == 0) return null;  
 E ret;  
 if(k == 0)  
 {  
 ret = first.element;  
 first = first.next;  
 size--;  
 return ret;  
 }  
 Node<E> newest = first.findNode(k-1);  
 ret = newest.next.element;  
 newest.next = newest.next.next;  
 size--;  
 return ret;  
}



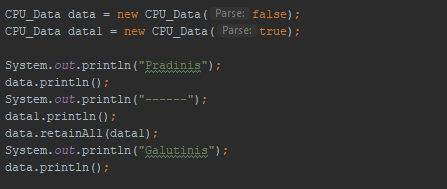


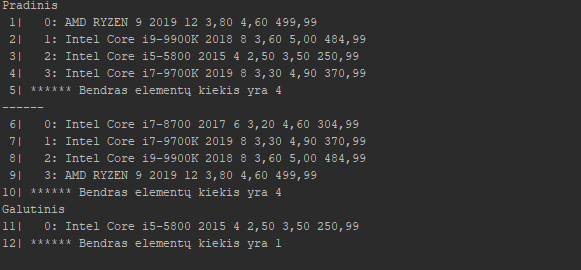
@Override  
public E set(int k, E e) {  
 if(size == 0 || e == null || k >= size) return null;  
   
 Node<E> currentnode = first.findNode(k);  
 E old = currentnode.element;  
 currentnode.element = e;  
 return old;  
}



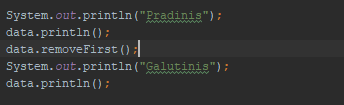


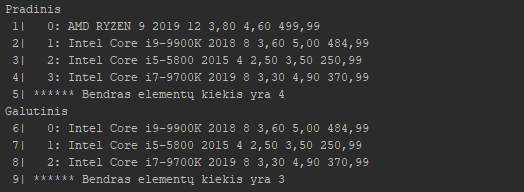
public boolean retainAll(LinkedList<E> c)  
{  
 boolean Loop = true;  
 Node<E> node = first;  
 Node<E> node1 = c.first;  
 while(Loop)  
 {  
 //System.out.println("" + node.element + " " + node1.element);  
 if(node.element.compareTo(node1.element) == 0)  
 {  
 removeFirstOccurency(node.element);  
 if(first == null)  
 {  
 return true;  
 }  
 node1 = c.first;  
 node = first;  
 }  
 else if(node1.next == null)  
 {  
 if(node.next != null) {  
 node = node.next;  
 node1 = c.first;  
 }  
 else  
 {  
 return true;  
 }  
 }  
 else {  
 node1 = node1.next;  
 }  
 }  
 return false;  
}



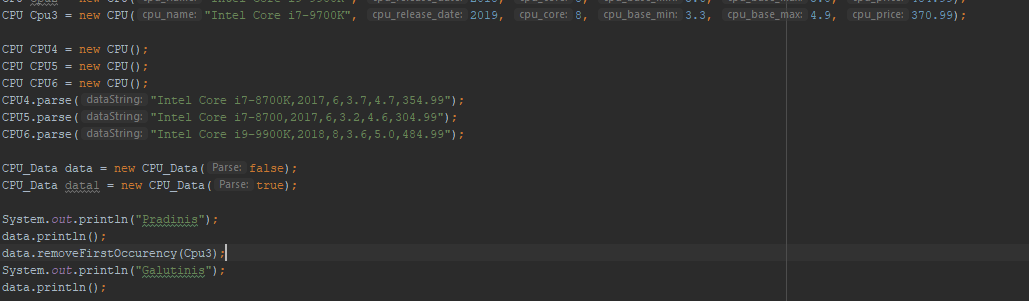


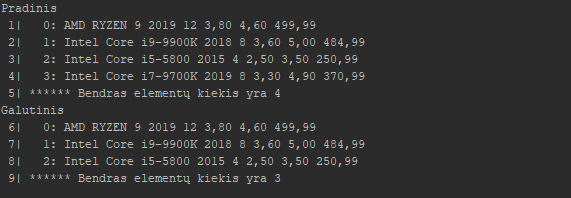
public E removeFirst()  
{  
 if(size == 0) return null;  
 Node<E> currentnode = first;  
 E removed = first.element;  
 switch (size)  
 {  
 case 1:  
 {  
 first = null;  
 size--;  
 return removed;  
 }  
 case 2:  
 {  
 currentnode.element = currentnode.next.element;  
 currentnode.next = null;  
 size--;  
 return removed;  
 }  
 default:  
 {  
 currentnode.element = currentnode.next.element;  
 currentnode.next = currentnode.next.next;  
 size--;  
 return removed;  
 }  
 }  
}





public boolean removeFirstOccurency(Object obj)  
{  
 if(size == 0 || obj == null) return false;  
  
 if(size == 1)  
 {  
 if(first.element == (E) obj) {  
 removeFirst();  
 return true;  
 }  
 }  
 for(int i =0; i < size; i++) {  
 Node<E> node = first.findNode(i);  
 if (i == 0 && node.element.compareTo((E) obj) == 0) {  
 node.element = node.next.element;  
 node.next = node.next.next;  
 size--;  
 return true;  
 }  
 if(node.next.next == null && node.next.element.compareTo((E) obj) == 0)  
 {  
 node.next.element = null;  
 node.next = null;  
 size--;  
 return true;  
 }  
 else if(node.next.element.compareTo((E) obj) == 0)  
 {  
 node.next.element = node.next.next.element;  
 node.next = node.next.next;  
 size--;  
 return true;  
 }  
 }  
 return false;  
}





**Greitaveikos rezultatai:**

