Disenjimi sipas abstraksionit

Faton Berisha

Departamenti i matematikës
Fakulteti i Shkencave Matematike-natyrore
Universiteti i Prishtinës

Qëllimet dhe objektivat

- Disenjim komponenesh të rishfrytëzueshme dhe fleksibile duke përdorur klasa abstrakte, interfejsa dhe shabllona disenji.
- Prezantim i shabllonave të disenjit për abstraksion:
  - Metodë shabllon (Template Method)
  - Strategji (Strategy)
  - Iterator
  - Factory

Përmbajtja

Shabllona disenji

- Koncepti i *shabllonave të disenjit* u përdor për të parën herë (Ch. Alexander, 1979) për të përshkruar disenje arkitektonike.
- Një shabllon disenji paraqet një zgjidhje gjenerike (d.m.th., të rishfrytëzueshme) për një problem që përsëritet.
- Disenjimi i softuerit i përngjet disenjimit arkitektonik.

Shabllona disenji. (Vazhdimi)

- Punën e pionierit mbi shabllonet e disenjit të softuerit e kanë bërë E. Gamma et al. (1995).
- Kompiluan 23 shabllone disenji më së shpeshti të përdorura me destinim të përgjithshëm që janë të pavarura nga domeni i aplikacionit dhe i klasifikuan në tri kategoritë vijuese.
  1. *Shabllona krijimi (creational patterns)*
  2. *Shabllona strukture (structural patterns)*
  3. *Shabllona sjelljeje (behaviorial patterns)*

Shembull shablloni disenji: Klasa Singleton

- Shablloni i disenjit *Singleton*
- Kategoria: Shabllon disenji krijimi
- Zbatueshmëria: Shfrytëzoni shabllonin Singleton
  kur duhet të ekzistojë saktësisht një instancë e një klase
  dhe duhet të jetë me qasje për klientët
  nga një pikë qasjeje publike
- Shembuj zbatimi: Dritarja e nivelit të parë e një aplikacioni,
  kohëmatësi i tërë një sistemi.

Shabllona disenji

```
public class Singleton {
  private static Singleton theInstance = null;

  public static Singleton getInstance() {
    if (theInstance = null) {
      theInstance = new Singleton();
    }
    return theInstance;
  }

  protected Singleton() {
    //inicializimi i fushave të instancës
  }
  //metodat dhe fushat tjera
}
```

Disenjimi i komponenteve gjenerike

- *Komponentë gjenerike (e rishfytëzueshme):* Komponentë programi, zakonisht klasë ose pako, që mund të adaptohet dhe shfrytëzohet në shumë kontekste të ndryshme pa modifikuar kodin e saj burimor.
- Teknikat themelore për disenjimin e komponentave gjenerike:
  - faktorizimi
  - gjeneralizimi (përgjithësimi)
- Mekanizmat për ndërtimin e komponenteve gjenerike:
  - inheritimi
  - delegimi

Faktorizimi

- *Faktorizimi* konsiston nga:
  - Identifikimi i segmentesh kodi në program që implementojnë të njëjtën logjikë, shpesh me saktësisht të njëjtin kod, në shumë vende të ndryshme
  - Kapja e logjikës në një komponentë gjenerike e cila definohet njëherë
  - Riorganizimi i programit ashtu që secila paraqitje e segmentit të kodit zëvendësohet me përdorimin e komponentës gjenerike.

Faktorizimi

Udhëzues disenji: Faktorizoni segmentet e përsëritura të kodit

Segmente të përsëritura kodi të mbështetura në të njëjtën logjikë krijojnë rrezik për mirëmbajtje.
Ato duhet të faktorizohen ashtu që segmenti i kodit
të paraqitet vetëm njëherë.

Faktorizimi me anë të invokimit të një metode

- Forma më e thjeshtë e faktorizimimit mund të bëhet përmes invokimit të një metode.
- Shembull kodi me segmente të përsëritura:

```
class Computation {
  void method1(...) {
    //...
    computeStep1();
    computeStep2();
    computeStep3();
    //...
  }
```

Faktorizimi me anë të invokimit të një metode. (Vazhdim)

- ...

```
  void method2(...) {
    //...
    computeStep1();
    computeStep2();
    computeStep3();
    //...
  }
  //...
}
```

Faktorizimi me anë të invokimit të një metode. (Vazhdim)

- Faktorizimi me anë të një metode:

```
class FactorizedComputation {
  void computeAll(...) {
    computeStep1();
    computeStep2();
    computeStep3();
  }
```

Faktorizimi me anë të invokimit të një metode. (Vazhdim)

- ...

```
  void method1(...) {
    //...
    computeAll();
    //...
  }
  void method2(...) {
    //...
    computeAll();
    //...
  }
  //...
}
```

Faktorizimi me inheritim

- Faktorizimi i segmentesh të përsëritura kodi mund të bëhet me inheritim.
- Shembull kodi me segmente të përsëritura:

```
class Computation1 {
  void method1(...) {
    //...
    computeStep1();
    computeStep2();
    computeStep3();
    //...
  }
  //...
}
```

Faktorizimi me inheritim. (Vazhdim)

- ...

```
class Computation2 {
  void method2(...) {
    //...
    computeStep1();
    computeStep2();
    computeStep3();
    //...
  }
  //...
}
```

Faktorizimi me inheritim. (Vazhdim)

- Futja e në superklase të përbashkët:

```
class Common {
  void computeAll(...) {
    computeStep1();
    computeStep2();
    computeStep3();
  }
}
```

Faktorizimi me inheritim. (Vazhdim)

- Faktorizimi me inheritim:

```
class Computation1 extends Common {
  void method1(...) {
    //...
    computeAll();
    //...
  }
  //...
}
```

Faktorizimi me inheritim. (Vazhdim)

- ...

```
class Computation2 extends Common {
  void method2(...) {
    //...
    computeAll();
    //...
  }
  //...
}
```

Faktorizimi me delegim

- Faktorizimi i segmentesh të përsëritura kodi mund të bëhet me delegim.
- Shembull paraprak i kodit me segmente të përsëritura.
- Futja e në klase ndihmëse:

```
class Helper {
  void computeAll(...) {
    computeStep1();
    computeStep2();
    computeStep3();
  }
}
```

Faktorizimi me delegim. (Vazhdim)

- Faktorizimi me delegim:

```
class Computation1 {
  Helper helper;
  //...
  void method1(...) {
    //...
    helper.computeAll();
    //...
  }
  //...
}
```

Faktorizimi me delegim. (Vazhdim)

- ...

```
class Computation2 {
  Helper helper;
  //...
  void method2(...) {
    //...
    helper.computeAll();
    //...
  }
  //...
}
```

Aplet animimi gjenerik

```
import java.awt.*;
import javax.swing.*;
public class AnimationApplet extends JApplet
                             implements Runnable {
  protected Thread animationThread;
  protected int delay = 100;

  public void start() {
    animationThread = new Thread(this);
    animationThread.start();
  }
  public void stop() {
    animationThread = null;
  }
```

Aplet animimi gjenerik. (Vazhdim)

```
   public void run() {
     while (Thread.currentThread() == animationThread) {
         sleep();
         repaint();
     }
   }
   public void sleep() {
     try {
         Thread.sleep(delay);
     } catch (InterruptedException e) {
     }
   }
  final public void setDelay(int delay) {
    this.delay = delay;
  }
  final public int getDelay() {
    return delay;
  }
}
```

Aplet animimi gjenerik. (Vazhdim)

```
import java.awt.*;
import java.util.*;
public class DigitalClock3 extends AnimationApplet {
  protected Dimension d;
  protected Font font = new Font("Monospaced", Font.BOLD, 48);
  protected Color color = Color.green;

  public DigitalClock3() {
    setDelay(1000);
  }
  public void init() {
    d = getSize();
  }
```

Aplet animimi gjenerik. (Vazhdim)

```
  public void paint(Graphics g) {
    g.setColor(getBackground());
    g.fillRect(0, 0, d.width, d.height);
    Calendar c = Calendar.getInstance();
    int hour = c.get(Calendar.HOUR_OF_DAY);
    int minute = c.get(Calendar.MINUTE);
    int second = c.get(Calendar.SECOND);
    g.setFont(font);
    g.setColor(color);
    g.drawString(hour + ":" + minute / 10 + minute % 10
        + ":" + second / 10 + second % 10, 10, 60);
  }
}
```

Shablllon disenji: Metodë shablllon

- Shablloni i disenjit *Metodë shablllon*
- Kategoria: Shablllon sjelljeje
- Qëllimi: Defino skeletin e një algoritmi në një metodë, duke lënë disa hapa nënklasave, duke u lejuar kështu nënklasave të redefinojnë hapa të caktuar të algoritmit
- Zbatueshmëria:
  - Të implementohen pjesët e pandryshueshme të një algoritmi një herë dhe t'i lehen nënklasave sjellja që mund të ndryshojë
  - Të faktorizohet dhe lokalizohet sjellja e përbashkët ndërmjet nënklasave që të eliminohet duplifikimi i kodit

Një klasë gjenerike për grafik funksioni

```
import java.awt.*;
import javax.swing.*;

public abstract class Plotter extends JApplet {
    protected Dimension d;
    protected Color color = Color.black;
    protected int xorigin, yorigin;
    protected int xratio = 100, yratio = 100;

    public abstract double func(double x);
```

Një klasë gjenerike për grafik funksioni (Vazhdim)

```
public void init() {
    d = getSize();
    String att = getParameter("xratio");
    if (att != null)
        xratio = Integer.parseInt(att);
    att = getParameter("yratio");
    if (att != null)
        yratio = Integer.parseInt(att);
    att = getParameter("xorigin");
    if (att != null)
        xorigin = Integer.parseInt(att);
    else
        xorigin = d.width / 2;
    att = getParameter("yorigin");
    if (att != null)
        yorigin = Integer.parseInt(att);
    else
        yorigin = d.height / 2;
}
```

Një klasë gjenerike për grafik funksioni (Vazhdim)

```
public void paint(Graphics g) {
//  d = getSize();
    drawCoordinates(g);
    plotFunction(g);
}

protected void plotFunction(Graphics g) {
    for (int px = 0; px < d.width; px++) {
        try {
            double x = (double) (px - xorigin) / (double) xratio;
            double y = func(x);
            int py = yorigin - (int) (y * yratio);
            g.fillOval(px - 1, py - 1, 3, 3);
        } catch (Exception e) {
        }
    }
}
```

Një klasë gjenerike për grafik funksioni (Vazhdim)

```
protected void drawCoordinates(Graphics g) {
    g.setColor(Color.white);
    g.fillRect(0, 0, d.width, d.height);

    g.setColor(color);
    g.drawLine(0, yorigin, d.width, yorigin);
    g.drawLine(xorigin, 0, xorigin, d.height);

    g.setFont(new Font("TimeRoman", Font.PLAIN, 10));
    int px, py;
    int i = 1;
    py = yorigin + 12;
    g.drawString("0", xorigin + 2, py);
    for (px = xorigin + xratio; px < d.width; px += xratio) {
        g.drawString(Integer.toString(i++), px - 2, py);
        g.drawLine(px, yorigin - 2, px, yorigin + 2);
    }

    i = -1;
    for (px = xorigin - xratio; px >= 0; px -= xratio) {
        g.drawString(Integer.toString(i--), px - 2, py);
        g.drawLine(px, yorigin - 2, px, yorigin + 2);
    }
```

Një klasë gjenerike për grafik funksioni (Vazhdim)

```
        i = 1;
        px = xorigin + 4;
        for (py = yorigin - yratio; py >= 0; py -= yratio) {
            g.drawString(Integer.toString(i++), px, py + 4);
            g.drawLine(xorigin - 2, py, xorigin + 2, py);
        }

        i = -1;
        for (py = yorigin + yratio; py < d.height; py += yratio) {
            g.drawString(Integer.toString(i--), px, py + 4);
            g.drawLine(xorigin - 2, py, xorigin + 2, py);
        }

    }

}
```

Një klasë gjenerike për grafik funksioni (Vazhdim)

```
public class PlotSine extends Plotter {
    public double func(double x) {
        return Math.sin(x);
    }
}

public class PlotSqrt extends Plotter {
    public double func(double x) {
        return Math.sqrt(x);
    }
}
```

Një klasë gjenerike për grafik funksioni (Vazhdim)

```
public class PlotLog extends Plotter {
    public double func(double x) {
        return Math.log(x);
    }
}

public class PlotCosine extends Plotter {
    public double func(double x) {
        return Math.cos(x);
    }
}
```

Përgjithësimi

- *Përgjithësimi* është proces që merr një zgjidhje të një problemi specifik dhe e riorganizon ashtu që të zgjidhë jo vetëm problmin origjinal por gjithashtu edhe një kategori problemesh që janë të ngjashme me problemin origjinal.
- Shembull: Një klasë gjenerike për paraqitje grafike të disa funksionesh

Një klasë gjenerike për grafikë të shumëfishtë

```
public interface Function {
   double value(double x);
}

public class Sine implements Function {
  public double value(double x) {
    return Math.sin(x);
  }
}

public class Cosine implements Function {
  public double value(double x) {
    return Math.cos(x);
  }
}
```

Një klasë gjenerike për grafikë të shumëfishtë (Vazhdim)

```
import java.awt.*;
public abstract class MultiPlotter extends Plotter {
  protected static int MAX_FUNCTIONS = 5;
  protected int numOfFunctions = 0;
  protected Function functions[] = new Function[MAX_FUNCTIONS];
  protected Color colors[] = new Color[MAX_FUNCTIONS];

  abstract public void initMultiPlotter();

  public void init() {
    super.init();
    initMultiPlotter();
  }
```

Një klasë gjenerike për grafikë të shumëfishtë (Vazhdim)

```
final public void addFunction(Function f, Color c) {
  if (numOfFunctions < MAX_FUNCTIONS && f  !=  null) {
    functions[numOfFunctions] = f;
    colors[numOfFunctions] = c;
    numOfFunctions++;
  }
}
```

Një klasë gjenerike për grafikë të shumëfishtë (Vazhdim)

```
protected void plotFunction(Graphics g) {
  for (int i = 0; i < numOfFunctions; i++) {
     if (functions[i] != null) {
       Color c = colors[i];
       if (c != null) {
         g.setColor(c);
        } else {
          g.setColor(Color.black);
        }
        for (int px = 0; px < d.width; px++) {
          try {
            double x = (double) (px - xorigin) / (double) xratio;
            double y = functions[i].value(x);
            int py = yorigin - (int) (y * yratio);
            g.fillOval(px - 1, py - 1, 3, 3);
          } catch (Exception  e) {}
         }
     }
  }
}
```

Një klasë gjenerike për grafikë të shumëfishtë (Vazhdim)

```
  public double func(double x) {
    return 0.0;
  }
}
```

Një klasë gjenerike për grafikë të shumëfishtë (Vazhdim)

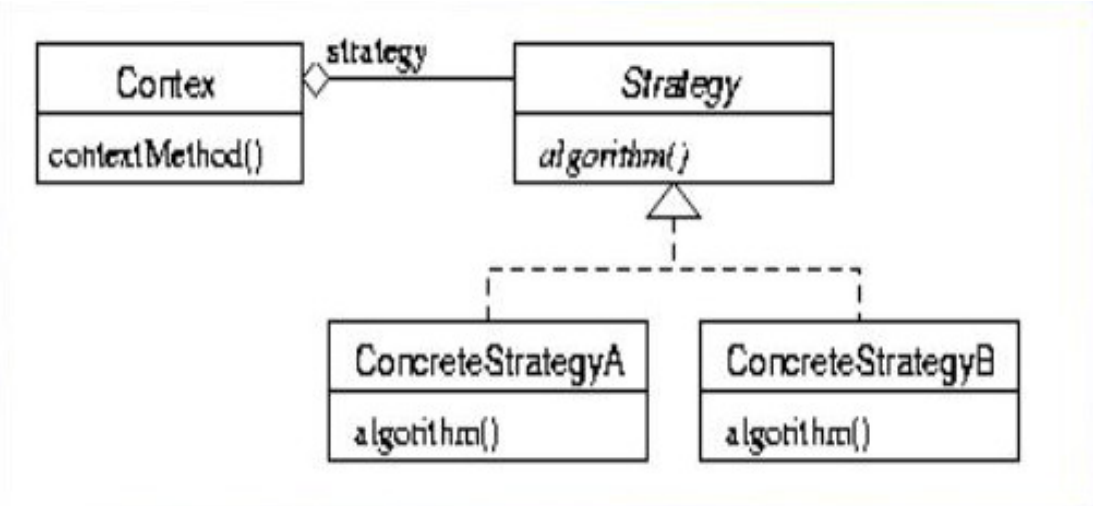
```
import java.awt.*;

public class PlotSineCosine extends MultiPlotter {
  public void initMultiPlotter() {
    addFunction(new Sine(), Color.green);
    addFunction(new Cosine(), Color.blue);
  }
}
```

Shablon disenji: Strategy

- Shablloni i disenjit *Strategy*
- Kategoria: Shablon sjelljeje
- Qëllimi: Defino një familje algoritmash, enkapsulo secilin dhe bëji ndërsjellazi të zëvendësueshëm
- Zbatueshmëria kur:
  - Shumë klasa të ndërlidhura ndryshojnë vetëm për nga sjellja e tyre
  - Kërkohen varianta të ndryshme të të njëjtit algoritëm
  - Një algoritëm shfrytëzon të dhëna për të cilat nuk duhet të dijnë klientët
  - Një klasë definon shumë sjellje, të cilat paraqiten si urdhëra të shumfishtë kondicionalë në metodat e saj

Struktura e shabllonit të disenjit Strategy



Lidhja abstrakte

- *Lidhja abstrakte* i referohet mënyrës së lidhjes së klientëve me ofruesit e servisit.
- Një klient i qaset një servisi përmes një interfejsi ose një klase abstrakte pa e ditur klasën aktuale konkrete që ofron servisin.
- Shablloni i disenji Strategy është një shembull i lidhjes abstrakte.

Programo sipas interfejsi, jo sipas implementimi

Udhëzues disenji: Programo sipas interfejsi, jo sipas implementimi

Veço interfejsin nga implementimi.
Klientët e një klase duhet qasur vetëm funksionaliteteve të klasës përmes interfejsit të saj.
Implementimi duhet të jetë i fshehur dhe irrelevant për klientin.

Iterimi nëpër koleksione

```
public class Course {
    public String dept, code, title;
    public int level;

    public Course(String dept, String code, String title,
                  int level) {
        this.dept = dept;
        this.code = code;
        this.title = title;
        this.level = level;
    }

    public String toString() {
        return dept + " " + code + " " + title;
    }
```

Iterimi nëpër koleksione (Vazhd.)

```
    public boolean isPrerequisiteOf(Course other) {
        boolean answer = false;
        if (other != null && dept.equals(other.dept)
                    && level < other.level) {
            answer = true;
        }
        return answer;
    }
}
```

Zgjidhja A: Qasja direkte

```
LinkedList list;
//... Popullimi i listes me kurse
for (LinkedList.Node curr = list.head; curr != null;
     curr = curr.next) {
    System.out.println(curr.element);
}
```

Zgjidhja B: Itero përmes invokimi metode

```
public class IterList extends LinkedList {
    protected Node curr;

    public void reset() {
        curr = head;
    }

    public Object next() {
        Object obj = null;
        if (curr != null) {
            obj = curr.element;
            curr = curr.next;
        }
        return obj;
    }
```

Zgjidhja B: Itero përmes invokimi metode (Vazhd.)

```
    public boolean hasNext() {
        return curr != null;
    }

}
```

Zgjidhja B: Itero përmes invokimi metode (Vazhd.)

```
LinkedList list;
//... Popullimi i listes me kurse
list.reset();
while (list.hasNext()) {
    System.out.println(list.next());
}
```

Zgjidhja B: Por, nuk funksionon zgjidhja...

```
list.reset();
while (list.hasNext()) {
    Course c1 = (Course) list.next();
    list.reset();
    while (list.hasNext()) {
        Course c2 = (Course) list.next();
        if (c1.isPrerequisiteOf(c2)) {
            System.out.println(c1
                + " është parakusht i " + c2);
        }
    }
}
```

Zgjidhja C: Ndaj iteratorin nga lista

```
public class LinkedListIterator {
    protected LinkedList.Node curr;
    protected LinkedList list;

    public LinkedListIterator(LinkedList list) {
        this.list = list;
        curr = list.head;
    }
```

Zgjidhja C: Ndaj iteratorin nga lista (Vazhd.)

```
    public Object next() {
        Object obj = null;
        if (curr != null) {
            obj = curr.element;
            curr = curr.next;
        }
        return obj;
    }

    public boolean hasNext() {
        return curr != null;
    }
}
```

Zgjidhja C: Tani funksionon...

```
LinkedListIterator i1 = new LinkedListIterator(list);
while (i1.hasNext()) {
    Course c1 = (Course) i1.next();
    LinkedListIterator i2 = new LinkedListIterator(list);
    while (i2.hasNext()) {
        Course c2 = (Course) i2.next();
        if (c1.isPrerequisiteOf(c2)) {
            System.out.println(c1
                + " është parakusht i " + c2);
        }
    }
}
```

Zgjidhja D: Një përgjithësim

```
public interface Iterator {
    public Object next();
    public boolean hasNext();
    void remove();
}

public interface List {
    public Iterator iterator();
    //...
}
```

Zgjidhja D: Një përgjithësim (Vazhd.)

```
public class LinkedList implements List {
    //...
    public Iterator iterator() {
        return new LinkedListIterator();
    }

    private class LinkedListIterator implements Iterator {
        private LinkedList.Node curr;

        public LinkedListIterator() {
            curr = head;
        }

        public boolean hasNext() {
            return curr != null;
        }
```

Zgjidhja D: Një përgjithësim (Vazhd.)

```
        public Object next() {
            Object obj = null;
            if (curr != null) {
                obj = curr.element;
                curr = curr.next;
            }
            return obj;
        }

        public void remove() {
            throw new UnSupportedOperationException();
        }
    }
}
```

Zgjidhja C: Iterimi me anë të iteratorëve abstraktë

```
List list = new LinkedList();
//... Popullimi i listës me kurse
Iterator i1 = list.iterator();
while (i1.hasNext()) {
    Course c1 = (Course) i1.next();
    Iterator i2 = list.iterator();
    while (i2.hasNext()) {
        Course c2 = (Course) i2.next();
        if (c1.isPrerequisiteOf(c2)) {
            System.out.println(c1
                + " është parakusht i " + c2);
        }
    }
}
```

Shabllon disenji: Iterator

- Shablloni i disenjit *Iterator*
- Kategoria: Shabllon sjelljeje
- Qëllimi: Ofro një mundësi qasjeje sekuenciale elementeve të një koleksioni
- Zbatueshmëria:
  - Qasja përmbajtjes së një koleksioni pa ekspozuar paraqitjen e brendshme të tij
  - Mbështjetja e iterimit të shumfishtë nëpër koleksione
  - Ofrimi i një interfejsi uniform për iterim nëpër koleksione të ndryshme (iterim polimorfik).

Shablloni i animimit me baferim të dyfishtë

```
import java.awt.*;

public abstract class DBAnimationApplet extends AnimationApplet {
    protected boolean doubleBuffered;
    protected Dimension d;
    protected Image im;
    protected Graphics offscreen;

    protected DBAnimationApplet(boolean doubleBuffered) {
        this.doubleBuffered = doubleBuffered;
    }
    protected DBAnimationApplet() {
        this.doubleBuffered = true;
    }
    public void run() {
        while (Thread.currentThread() == animationThread) {
            sleep();
            if (doubleBuffered) {
                render();
            } else {
                repaint();
            }
        }
    }
```

Shablloni i animimit me baferim të dyfishtë (Vazhdim)

```
    final public void render() {
        paintFrame(offscreen);
        Graphics g = getGraphics();
        g.drawImage(im, 0, 0, this);
    }
    final public void paint(Graphics g) {
        paintFrame(g);
    }
    final public void init() {
        d = getSize();
        im = createImage(d.width, d.height);
        offscreen = im.getGraphics();
        initAnimator();
    }
    protected void initAnimator() {
    }
    abstract protected void paintFrame(Graphics g);
}
```

Implementimi fillestar

- Implementimi fillestar i animimit të një algoritmi sortimi shfrytëzon apletin `DBAnimationApplet`.
- Klasa `AlgorithmAnimator` shfrytëzon shabllonin e disenjit Metodë shabllon.

Apleti gjenerik për animim algoritmi

```
public abstract class AlgorithmAnimator extends DBAnimationApplet {
    abstract protected void algorithm();

    public void run() {
        algorithm();
    }

    public void update() {
        sleep();
        if (doubleBuffered) {
            render();
        } else {
            repaint();
        }
    }
}
```

Animimi i sortimit

```
import java.awt.*;
public class Sort extends AlgorithmAnimator {
    int[] arr;
    String algName;

    protected void initAnimator() {
//      Dimension d = new Dimension(150, 150);
//      setSize(d);
        setDelay(20);
        algName = "BubbleSort";
//      algName = "QuickSort";
        String at = getParameter("alg");
        if (at != null)
            algName = at;
        scramble();
    }
    protected void scramble() {
        arr = new int[getSize().height / 2];
        for (int i = 0; i < arr.length;  i++) {
            arr[i] = i;
        }
        for (int i = arr.length; i > 0;  i--) {
            int j = (int) (i * Math.random());
            swap(arr, i - 1, j);
        }
    }
```

Animimi i sortimit (Vazhdim)

```
private void swap(int a[], int i, int j) {
    int t;
    t = a[i];
    a[i] = a[j];
    a[j] = t;
}
protected void paintFrame(Graphics g) {
    Dimension d = getSize();
    g.setColor(Color.white);
    g.fillRect(0, 0, d.width, d.height);
    g.setColor(Color.black);
    int y = 1;
    double f = d.width / (double) arr.length;
    for (int i = 0; i < arr.length; i++) {
        g.drawLine(0, y, (int) (arr[i] * f), y);
        y += 2;
    }
}
protected void bubbleSort(int a[]) {
    for (int i = a.length - 1; i > 0; i--)
        for (int j = 0; j < i; j++) {
            if (a[j] > a[j + 1])
                swap(a, j, j + 1);
            update();
        }
}
```

Animimi i sortimit (Vazhdim)

```
protected void quickSort(int a[], int lo0, int hi0) {
    int lo = lo0;
    int hi = hi0;
    int mid;
    update();
    if (hi0 > lo0) {
        mid = a[(lo0 + hi0) / 2];
        while (lo <= hi) {
            while ((lo < hi0) && (a[lo] < mid))
                ++lo;
            while ((hi > lo0) && (a[hi] > mid))
                --hi;
            if (lo <= hi) {
                swap(a, lo, hi);
                update();
                ++lo;
                --hi;
            }
        }
        if (lo0 < hi)
            quickSort(a, lo0, hi);
        if (lo < hi0)
            quickSort(a, lo, hi0);
    }
}
```

Animimi i sortimit (Vazhdim)

```
    protected void algorithm() {
        if ("BubbleSort".equals(algName))
            bubbleSort(arr);
        else if ("QuickSort".equals(algName))
            quickSort(arr, 0, arr.length - 1);
        else
            bubbleSort(arr);
    }
}
```

Ndarja e algoritmave

Udhëzues disenji: Veçoni funksionalitetet që adresojnë çështje të ndryshme

Në qoftë se një klasë përmban komponente që adresojnë çështje të ndryshme, këto komponente janë kandidatë për t'u veçuar nga klasa origjinale.

Algoritmi abstrakt i sortimit

```
public abstract class SortAlgorithm {
    private AlgorithmAnimator animator;

    protected SortAlgorithm(AlgorithmAnimator animator) {
        this.animator = animator;
    }
    abstract void sort(int a[]);
    protected void update() {
        if (animator != null)
            animator.update();
    }
    protected static void swap(int a[], int i, int j) {
        int t;
        t = a[i];
        a[i] = a[j];
        a[j] = t;
    }
}
```

Algoritmi konkret: BubbleSortAlgorithm

```
public class BubbleSortAlgorithm extends SortAlgorithm {
    void sort(int a[]) {
        for (int i = a.length - 1; i > 0; i--)
            for (int j = 0; j < i; j++) {
                if (a[j] > a[j + 1])
                    swap(a, j, j + 1);
                update();
            }
    }

    public BubbleSortAlgorithm(AlgorithmAnimator animator) {
        super(animator);
    }
}
```

Algoritmi konkret: QuickSortAlgorithm

```
public class QuickSortAlgorithm extends SortAlgorithm {
    public QuickSortAlgorithm(AlgorithmAnimator animator) {
        super(animator);
    }

    public void sort(int a[]) {
        qSort(a, 0, a.length - 1);
    }
```

Algoritmi konkret: `QuickSortAlgorithm` (Vazhdim)

```
    protected void qSort(int a[], int lo0, int hi0) {
        int lo = lo0;
        int hi = hi0;
        int mid;
        update();
        if (hi0 > lo0) {
            mid = a[(lo0 + hi0) / 2];
            while (lo <= hi) {
                while ((lo < hi0) && (a[lo] < mid))
                    ++lo;
                while ((hi > lo0) && (a[hi] > mid))
                    --hi;
                if (lo <= hi) {
                    swap(a, lo, hi);
                    update();
                    ++lo;
                    --hi;
                }
            }
            if (lo0 < hi)
                qSort(a, lo0, hi);
            if (lo < hi0)
                qSort(a, lo, hi0);
        }
    }
}
```

Instancimi i algoritmave konkretë të sortimit

- Instancimi i algoritmave konkretë të sortimit në metodën `initAnimator()` të klasës `Sort2` (që zgjeron `AlgorithmAnimator`) ka mangësi.
  - Të klientin (`Sort2`) e shabllonit të disenjit Strategy (`class SortAlgorithm`) të lidhur me strategjitë konkrete (`BubbleSortAlgorithm` dhe `QuickSortAlgorithm`).
  - Strategjitë duhet të jenë të këmbyeshme.
  - Klienti nuk duhet të jetë në dijeni mbi strategjitë konkrete.
- Alternativë më e mirë është përdorimi i një klase përgjegjësia e vetme e së cilës është krijimi i instancave të algoritmave konkretë të sortimit.
  - Një klasë e tillë quhet *factory*.

Factory abstrakt i algoritmit

```
public interface SortAlgorithmFactory {
    SortAlgorithm makeSortAlgorithm(String algName);
}
```

Factory konkret i algoritmit

```
public class StaticAlgorithmFactory implements SortAlgorithmFactory {
    protected AlgorithmAnimator animator;

    public StaticAlgorithmFactory(AlgorithmAnimator animator) {
        this.animator = animator;
    }
    public SortAlgorithm makeSortAlgorithm(String algName) {
        if ("BubbleSort".equals(algName))
            return new BubbleSortAlgorithm(animator);
        else if ("QuickSort".equals(algName))
            return new QuickSortAlgorithm(animator);
        else
            return new BubbleSortAlgorithm(animator);
    }
}
```

Animimi i reviduar i sortimit

```
import java.awt.*;
public class Sort2 extends AlgorithmAnimator {
    protected int arr[];
    protected String algName;
    protected SortAlgorithm theAlgorithm;
    protected SortAlgorithmFactory algorithmFactory;

    public void initAnimator() {
//      Dimension d = new Dimension(150, 150);
//      setSize(d);
        setDelay(20);
        algName = "BubbleSort";
//      algName = "QuickSort";
        String at = getParameter("alg");
        if (at != null)
            algName = at;
        algorithmFactory = new StaticAlgorithmFactory(this);
        theAlgorithm = algorithmFactory.makeSortAlgorithm(algName);
        scramble();
    }
```

Animimi i reviduar i sortimit

```
    protected void algorithm() {
        if (theAlgorithm != null)
            theAlgorithm.sort(arr);
    }
    protected void scramble() {
        arr = new int[getSize().height / 2];
        for (int i = 0; i < arr.length;  i++) {
            arr[i] = i;
        }
        for (int i = arr.length; i > 0;  i--) {
            int j = (int) (i * Math.random());
            SortAlgorithm.swap(arr, i - 1, j);
        }
    }
    protected void paintFrame(Graphics g) {
        Dimension d = getSize();
        g.setColor(Color.white);
        g.fillRect(0, 0, d.width, d.height);
        g.setColor(Color.black);
        int y = 1;
        double f = d.width / (double) arr.length;
        for (int i = 0; i < arr.length; i++) {
            g.drawLine(0, y, (int) (arr[i] * f), y);
            y += 2;
        }
    }
}
```

Shabllon disenji: Factory

- Shablloni i disenjit *Factory*
- Kategoria: Shabllon krijimi
- Qëllimi: Defino një interfejs për instancim objektesh por lejo nënklasat të vendosin cilën klasë ta instacojnë dhe si.
- Zbatueshmëria kur:
  - Një sistem duhet të jetë i pavarur nga ajo se si krijohen produktet e tij.

Udhëzime për lexim të mëtejmë

- `http://www.fberisha.org`
- X. Jia, *Object oriented software development using Java,* kapitulli 4.
- D. Schmidt, *Programming principles in Java: architectures and interfaces,* kapitulli 11.

Përfundim

- Shablllonat e disenjit janë përshkrime skematike të problemeve që përsëriten në disenjimin e softuerit.
- Komponentet gjenerike janë komponente, zakonisht klsas ose pako, që mund të adaptohen dhe shfrytëzohen në shumë kontekste të ndryshme pa modifikim kodi burimor.
- Udhëzues të rëndësishëm disenji:
  - Maksimizo fleksibilitetin
  - Minimizo gjasën e keqpërdorimit
  - Programo sipas interfejsit, jo sipas implementimit
  - Veço funksionalitetet që adresojnë çështje të ndryshme
  - Minimizo interfejsin