Komponentat dhe inheritimi

Faton Berisha

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

Qëllimet dhe objektivat

- Diskutimi i termave të lidhura me trashigiminë, si: mbingarkimi, inheritimi, mbikalimi, fshehja, nëntipet, polimorfizmi dhe ndryshimi i tipit (casting).
- Egzaminimi i disa çështjesh standarde të lidhura me disenjimin dhe implementimin e klasave dhe theksimi i disa parimeve disenjuese.
- Presantimi i disa apleteve më të sofistikuara animuese për të ilustruar teknikat në përdorim në animime.

Përmbajtja

Mbingarkimi i metodave dhe konstruktorëve

Mbingarkimi (Overloading)

- *Mbingarkim (overloading):* vetia e lejimit të metodave ose konstruktorëve të ndryshëm që të ndajnë të njëjtin emër.
- Themi se emri është *mbingarkuar* me implementime të shumëfishta.

Rregulla e mbingarkimit

Në qoftë se dy metoda ose konstruktorë në të njëjtën klasë kanë *nënshkrime* të ndryshme, atëherë mund të mbingarkohen në të njëjtin emër.

Mbingarkimi i metodave dhe konstruktorëve. (Vazhdim)

```
/** Modelon një pikë në rrafsh. */
public class Point {
  protected double x, y;
  /** Krijon origjinën (0,0). */
  public Point() {
    x = 0.0;
    y = 0.0;
  }
  /** Krijon pikën (x,y).
   * @param x koordinata horizontale
   * @param y koordinata vertikale */
  public Point(double x, double y) {
    this.x = x;
    this.y = y;
  }
```

Mbingarkimi i metodave dhe konstruktorëve. (Vazhdim)

```
/** Llogarit distancën ndërmjet kësaj pike dhe pikës tjetër.
 * @param other pika tjetër
 * @return distanca e llogaritur */
public double distance(Point other) {
  double dx = this.x - other.x;
  double dy = this.y - other.y;
  return Math.sqrt(dx * dx + dy * dy);
}
```

Mbingarkimi i metodave dhe konstruktorëve. (Vazhdim)

```
/** Llogarit distancën ndërmjet kësaj pike dhe (x,y).
 * @param x koordinata horizontale
 * @param y koordinata vertikale
 * @return distanca e llogaritur */
public double distance(double x, double y) {
  double dx = this.x - x;
  double dy = this.y - y;
  return Math.sqrt(dx * dx + dy * dy);
}
```

Mbingarkimi i metodave dhe konstruktorëve. (Vazhdim)

```
  /** Llogarit distancën ndërmjet kësaj pike
   *    dhe origjinës.
   * @return distanca e llogaritur */
  public double distance() {
    return Math.sqrt(x * x + y * y);
  }
}
```

Mbingarkimi i metodave dhe konstruktorëve. (Vazhdim)

```
public static void main(String[] args) {
  Point p1 = new Point(); //Invokon Point()
  //Invokon Point(double,double)
  Point p2 = new Point(20.0, 30.0);
  //Invokon distance(Point)
  System.out.println(p2.distance(p1));
  //Invokon distance(double,double)
  System.out.println(p2.distance(50.0, 60.0));
  System.out.println(p2.distance(50, 60)); //Sikur mësipër
  System.out.println(p2.distance()); //Invokon distance()
}
```

Mbingarkimi i metodave dhe konstruktorëve. (Vazhdim)

Udhëzues disenji: Shfrytëzoni arsyeshëm mbingarkimin

Mbingarkimi duhet të shfrytëzohet vetëm në dy situata:

1. kur ekziston një përshkrim i përgjithshëm, joveçues i funksionalitetit i cili u përgjigjet të gjitha metodave të mbingarkuara;
2. kur të gjitha metodat e mbingarkuara ofrojnë të njëjtin funksionalitet, disa prej të cilave ofrojnë argumentë të nënkuptuar.

Mbingarkimi i metodave dhe konstruktorëve. (Vazhdim)

Shembull i rastit të parë:

```
public class StringBuffer {
  public StringBuffer append(String str) { ... }
  public StringBuffer append(boolean b) { ... }
  public StringBuffer append(char c) { ... }
  public StringBuffer append(int i) { ... }
  public StringBuffer append(long l) { ... }
  public StringBuffer append(float f) { ... }
  public StringBuffer append(double d) { ... }
}
```

Përshkrimi: Paraqitja String e argumentit shtohet te fundi i përmbajtjes aktuale të string baferit

Mbingarkimi i metodave dhe konstruktorëve. (Vazhdim)

Shembull i rastit të dytë:

```
public class String {
  public String substring(int i, int j) { ... }
  public String substring(int i) { ... }
}
```

Funksionaliteti: Të dyja metodat kthejnë nënstring.

Zgjerimi i klasave

- Sintaksa:

```
[ ModifikatorëKlase ] class EmërKlase
    [ extends SuperKlasë ]
    [ implements Interfejs1,Interfejs2,...] {
  DeklarimeAnëtarëshKlase
}
```

- Semantika:
  - Relacioni i *inheritimit (trashigimisë)*;
  - Të gjithë anëtarët `public` dhe `protected` të superklasës mund të qasen në klasën e zgjeruar.

Zgjerimi i klasave. (Vazhdimi)

```
import java.awt.*;
/** Paraqet një pikë të ngjyrosur në rrafsh. */
public class ColoredPoint extends Point {
  private Color color;
  /** Krijon një pikë të ngjyrosur.
   * @param x koordinata horizontale
   * @param y koordinata vertikale
   * @param color ngjyra
   */
  public ColoredPoint(final double x, final double y,
                      final Color color) {
    super(x, y);
    this.color = color;
  }
```

Zgjerimi i klasave. (Vazhdimi)

```
  /** Krijon një pikë me ngjyrë të zezë.
   * @param x koordinata horizontale
   * @param y koordinata vertikale
   */
  public ColoredPoint(final double x, final double y) {
    this(x, y, Color.black);
  }

  /** Krijon origjinën me ngjyrë të zezë. */
  public ColoredPoint() {
    color = Color.black;
  }
}
```

Nëntipet dhe polimorfizmi

Nëntip

Tipi `T1` është *nëntip (subtype)* i tipit `T2` (*mbitip, supertype*) në qoftë se çdo vlerë legjitime e tipit `T1` është poashtu vlerë legjitime e tipit `T2`.

Rregulla e nëntipeve

Një vlerë e një nëntipi mund të paraqitet kudo që pritet vlerë e mbitipit të tij.

Nëntipet dhe polimorfizmi. (Vazhdim)

Rregulla e konvertimit të një tipi

- Konvertimi i një nëntipi në njërin nga mbitipet e tij është gjithmonë i lejuar dhe kryhet në mënyrë implicite kurdo që të jetë e nevojshme.
- Konvertimi i një mbitipi në njërin nga nëntipet e tij kërkon *konvertim eksplicit (explicit casting)* dhe gjithmonë lejohet gjatë kompilimit, por mund të rezultojë me përjashtim (run-time exception).

Nëntipet dhe polimorfizmi. (Vazhdim)

Rregulla e ndarjes polimorfike të vlerës

Tipi i shprehjes nga ana e djathtë e një ndarje vlere duhet të jetë nëntip i tipit të variablës nga ana e majtë e ndarjes së vlerës.

Nëntipet dhe polimorfizmi. (Vazhdim)

```
public class Student { ... }
public class Undergraduate extends Student { ... }
public class Graduate extends Student { ... }

//Ndarje polimorfike vlerash
Student student1 = new Undergraduate();
Student student2 = new Graduate();

Graduate student3 = student2; //Gabim!

Graduate student4 = (Graduate)student2;
student4 = (Graduate)student1; //Gabim!
```

Nëntipet dhe polimorfizmi. (Vazhdim)

Ekzistojnë dy mënyra të konvertimit korrekt në nëntip (*downcasting*):

1. Përdorimi i operatorit `instanceof`:

```
if (student1 instanceof Graduate) {
  Graduate gradStudent = (Graduate)student1;
  gradStudent.getResearchTopic();
}
```

2. Kapja e përjashtimit `ClassCastException`:

```
try {
  Graduate gradStudent = (Graduate)student1;
  gradStudent.getResearchTopic();
} catch (ClassCastException e) {
  //student1 nuk është Graduate
}
```

Mbikalimi i metodave

Mbikalimi (Overriding)

- *Mbikalimi (overriding)* i referohet futjes së një metode në nënklasë që ka të njëjtin emër, nënshkrim dhe tip kthyes sikur metoda në mbiklasë.
- Implementimi i metodës në nënklasën zëvendëson implementimin e metodës në mbiklasën.
- Një metodë e deklaruar si `final` nuk mund të mbikalohet.

Mbikalimi i metodave. (Vazhdim)

```
public class Student {
  protected String name;
  public Student(String name) {
    this.name = name;
  }
  public String toString() {
    return name;
  }
}
```

Mbikalimi i metodave. (Vazhdim)

```
public class Undergraduate extends Student {
  public Undergraduate (String name) {
    super(name);
  }
  public String toString() {
    return "Studenti Bachelor: " + name;
  }
}
```

Mbikalimi i metodave. (Vazhdim)

```
public class Graduate extends Student {
  public Graduate (String name) {
    super(name);
  }
  public String toString() {
    return "Studenti Master: " + name;
  }
}
```

Mbikalimi i metodave. (Vazhdim)

```
public class Course {
  protected static final int CAPACITY = 40;
  protected Student[] student = new Student[CAPACITY];
  protected int count = 0;
  public void enroll(Student s) {
    if (s != null && count < CAPACITY) {
      student[count++] = s;
    }
  }
  public void list() {
    for (int i = 0; i < count; i++) {
      System.out.println(student[i].toString());
    }
  }
}
```

Mbikalimi i metodave. (Vazhdim)

Figura: Diagrami i klasave të studentëve dhe kurseve

Mbikalimi i metodave. (Vazhdim)

```
public class TestCourse {
  public static void main(String[] args) {
    Course c = new Course();
    c.enroll(new Undergraduate("Filan"));
    c.enroll(new Graduate("Tringa"));
    c.enroll(new Undergraduate("Fistek"));
    c.list();
  }
}
```

Invokimi i metodave të mbikaluara

```
public class Point {
  //...
  /** Krahason në barazim këtë objekt me një objekt tjetër.
   * @param other objekti tjetër
   * @return true nëse të barabartë, përndryshe false*/
  public boolean equals(Object other) {
    boolean answer = false;
    if (other != null && other instanceof Point) {
      Point p = (Point)other;
      answer = (x == p.x && y == p.y);
    }
    return answer;
  }
}
```

Invokimi i metodave të mbikaluara. (Vazhdim)

```
public class ColoredPoint extends Point {
  //...
  /** Krahason në barazim këtë objekt me një objekt tjetër.
   * @param other objekti tjetër
   * @return true nëse të barabartë, përndryshe false*/
  public boolean equals(Object other) {
    boolean answer = false;
    if (other != null && other instanceof ColoredPoint) {
      ColoredPoint p = (ColoredPoint)other;
      answer = super.equals(p) && color.equals(p.color);
    }
    return answer;
  }
}
```

Invokimi i metodave të mbikaluara. (Vazhdim)

```
Point p2 = new Point(20.0, 30.0);
Point p4 = new ColoredPoint(20.0, 30.0);

System.out.println(p2.equals(new Point(20, 30))); //true
System.out.println(p4.equals(p2)); //false
System.out.println(p2.equals(p4)); //true
```

Sërish mbi nëntipet

Relacionet nëntip

- Në qoftë se `class C1 extends C2`, atëherë `C1 <= C2`.
- Në qoftë se `interface I1 extends I2`, atëherë `I1 <= I2`.
- Në qoftë se `class C implements I`, atëherë `C <= I`.
- Për çdo `class C` dhe `interface I`, `C <= Object` dhe `I <= Object`.
- Për çdo tip reference ose primitiv `T`, `T[] <= Object`.
- Në qoftë se `T1 <= T2`, atëherë `T1[] <= T2[]`.

Implementimi i interfjesave

```
public interface StudentBehavior {
  public float getGPA();
}

public interface EmployeeBehavior {
  public float getSalary();
}
```

Implementimi i interfjesave. (Vazhdim)

```
public class Student implements Student Behavior {
  protected float gpa;
  // ... fushat dhe metodat tjera
  public float getGPA() {
    return gpa;
  }
}

public class Employee implements EmployeeBehavior {
  protected float salary;
  // ... fushat dhe metodat tjera
  public float getSalary() {
    return salary;
  }
}
```

Implementimi i interfjesave. (Vazhdim)

```
public class StudentEmployee
      implements StudentBehavior, EmployeeBehavior {
  protected float gpa;
  protected float salary;
  // ... metodat dhe fushat tjera
  public float getGPA() {
    return gpa;
  }
  public float getSalary() {
    return salary;
  }
}
```

Implementimi i interfjesave. (Vazhdim)

Figura: Implementimi i shumëfishtë i interfejsave

Trashigimia dhe implementimi

Zgjerimi dhe implementimi

- Implementimi i shumëfishtë i interfejsave është legal në Java.
- Lejohet zgjerimi i një klase dhe implementimi i disa interfejsave.
- Por, zgjerimi i shumëfishtë i klasave nuk lejohet në Java.

Trashigimia dhe implementimi. (Vazhdim)

Si të zgjidhet problemi i nevojës së zgjerimit të shumfishtë?

Figura: Diagrami i klasave të studentëve dhe kurseve

Implementimi i interfjesave. (Vazhdim)

```
public class StudentEmployee2
      implements StudentBehavior, EmployeeBehavior {
  protected Student student;
  protected Employee employee;
  public StudentEmployee2(String name) {
    student = new Student(name);
    employee = new Employee(name);
  }
  public float getGPA() {
    return student.getGPA();
  }
  public float getSalary() {
    return employee.getSalary();
  }
}
```

Fshehja e fushave dhe metodave statike

Fshehja

Fshehja i referohet deklarimit të një fushe ose metode statike në një nënklasë që ka të njëjtin emër sikurse një fushë ose metodë statike në superklasën.

Mbikalimi dhe fshehja

Mbikalimi dhe fshehja janë koncepte të ndryshme:

- Metodat e instancës vetëm mund të mbikalohen.
- Metodat statike dhe fushat vetëm mund të fshihen.

Fshehja e fushave dhe metodave statike. (Vazhdim)

```
public class Point {
  private static String className = "Point";
  public static String getDescription() {
    return className;
  }
  // deklarimet tjera
}

public class ColoredPoint extends Point {
  private static String className = "ColoredPoint";
  public static String getDescription() {
    return className + " e ngjyrosur";
  }
  // deklarimet tjera
}
```

Fshehja e fushave dhe metodave statike. (Vazhdim)

```
ColoredPoint p5 = new ColoredPoint(20.0, 30.0, Color.black);
Point p6 = p5;

System.out.println(p5.getDescription()); //ColoredPoint
                                         //   e ngjyrosur
System.out.println(p6.getDescription()); //Point
```

Fshehja e fushave dhe metodave statike. (Vazhdim)

Udhëzues disenji: Evitoni fshehjen

- Evitoni fshehjen e fushave dhe metodave statike; përdorni emra të ndryshëm për gjëra të ndryshme.
- Invokoni metodat statike me anë të emrave të klasave, e jo me anë të referencimit të objekteve.

Fshehja e fushave dhe metodave statike. (Vazhdim)

```
System.out.println(ColoredPoint.getDescription());
System.out.println(Point.getDescription());
```

Evitimi i fushave publike

Udhëzues disenji: Evitoni fushat publike

Nuk duhet të ketë fusha publike jofinale,
përveç në qoftë se klasa është finale
dhe fusha është e pakushtëzuar.

Evitimi i fushave publike. (Vazhdim)

```
public class Point {
  protected double x, y;
  //deklarimet tjera
 /** Krijon origjinën (0,0). */
 public Point() {
 x = 0.0;
 y = 0.0;
 }
```

Evitimi i fushave publike. (Vazhdim)

```
/** Kthen koordinatën horizontale të kësaj pike.
 * @return koordinata horizontale */
public double getX() {
  return x;
}
/** Vë koordinatën horizontale të kësaj pike.
 * @param x koordinata horizontale */
public void setX(final double x) {
  this.x = x;
}
```

Evitimi i fushave publike. (Vazhdim)

```
  /** Kthen koordinatën vertikale të kësaj pike.
   * @return koordinata vertikale */
  public double getY() {
    return y;
  }

  /** Vë koordinatën vertikale të kësaj pike.
   * @param y koordinata vertikale */
  public void setY(final double y) {
    this.y = y;
  }
}
```

Evitimi i fushave publike. (Vazhdim)

```
/** Modelon një pikë në koordinata polare. */
public class PolarPoint extends Point {
  protected double angle;
  protected double radius;
  //Invariantë: x = angle * Math.cos(radius)
  //Invariantë: y = angle * Math.sin(radius)
  /** Konstrukton origjinën. */
  public PolarPoint() {
    super();
    radius = 0;
  }
```

Evitimi i fushave publike. (Vazhdim)

```
/** Konstrukton pikën.
 * @param angle këndi në radian
 * @param radius rrezja */
public PolarPoint(double angle, double radius) {
  this.angle = angle;
  this.radius = radius;
  polarToRectangular();
}
```

Evitimi i fushave publike. (Vazhdim)

```
/** Kthen këndin.
 * @return këndi */
public double getAngle() {
  return angle;
}
/** Vë këndin.
 * @param angle këndi */
public void setAngle(double angle) {
  this.angle = angle;
  polarToRectangular();
}
```

Evitimi i fushave publike. (Vazhdim)

```
/** Kthen rrezen.
 * @return rrezja */
public double getRadius() {
  return radius;
}
/** Vë rrezen.
 * @param radius rrezja */
public void setRadius(double radius) {
  this.radius = radius;
  polarToRectangular();
}
private void polarToRectangular() {
  x = radius * Math.cos(angle);
  y = radius * Math.sin(angle);
}
```

Evitimi i fushave publike. (Vazhdim)

```
  /** Vë koordinatën horizontale.
   * @param x koordinata horizontale */
  public void setX(double x) {
    this.x = x;
    rectangularToPolar();
  }
  /** Vë koordinatën vertikale.
   * @param y koordinata vertikale */
  public void setY(double y) {
    this.y = y;
    rectangularToPolar();
  }
  private void rectangularToPolar() {
    angle = Math.atan2(y, x);
    radius = Math.sqrt(x * x + y * y);
  }
}
```

Veçimi i interfejsit nga implementimi

Udhëzues disenji: Veçoni interfejsin nga implementimi

Kur funksionaliteti i ofruar nga një klasë
mund të implementohet në mënyra të ndryshme,
është e këshillueshme të veçohet interfejsi nga implementimi.

Mbani mend!

Programo sipas interfejsit, jo sipas implementimit!

Veçimi i interfejsit nga implementimi. (Vazhdim)

```
/** Specifikon një listë. */
public interface List {
  /** Kthen numrin e elementeve të listës.
   * @return numri i elementeve */
  public int getCount();
  /** Kthen se a është lista boshe.
   * @return true nëse boshe, false përndryshe */
  public boolean isEmpty();
```

Veçimi i interfejsit nga implementimi. (Vazhdim)

```
/** Kthen elementin e listës në pozitën e specifikuar.
 * @param index indeksi i pozitës
 * @return elementi në pozitën e specifikuar */
public Object get(int index);
/** Kthen elementin në kreun e listës.
 * @return elementi në kreun */
public Object getHead();
/** Kthen elementin në bishtin e listës.
 * @return elementi në bishtin */
public Object getTail();
```

Veçimi i interfejsit nga implementimi. (Vazhdim)

```
/** Inserton elementin në pozitën e dhënë në listën.
 * @param index indeksi i pozitës
 * @param element elementi i specifikuar */
public void add(int index, Object element);
/** Inserton elementin kreun e listës.
 * @param element elementi i specifikuar */
public void addHead(Object element);
/** Inserton elementin bishtin e listës.
 * @param element elementi i specifikuar */
public void addTail(Object element);
```

Veçimi i interfejsit nga implementimi. (Vazhdim)

```
  /** Fshin elementin në pozitën e dhënë në listën.
   * @param index indeksi i pozitës
   * @return elementi i fshirë */
  public Object remove(int index);
  /** Fshin elementin në kreun e listës.
   * @return elementi i fshirë */
  public Object removeHead();
  /** Fshin elementin në bishtin e listës.
   * @return elementi i fshirë */
  public Object removeTail();
}
```

Veçimi i interfejsit nga implementimi. (Vazhdim)

```
public class LinkedList implements List {
  //Trupi
}

public class ArrayList implements List {
  //Trupi
}
```

Organizimi i fajlave

Organizimi i klasave në fajla

Ekzistojnë dy lloje klasash:

- Klasat për shfrytëzim të përgjithshëm
  - Deklarohen publike.
  - Ruhen në fajl emri i të cilit përputhet me emrin e klasës.
  - P.sh., `public class LinkedList`
- *Klasat ndihmëse,* të cilat shfrytëzohen vetëm për të implementuar klasa tjera
  - Nuk deklarohen publike.
  - Nuk ruhen në fajla të veçantë.
  - P.sh., `class Node`

Organizimi i fajlave. (Vazhdim)

Ekzistojnë dy zgjidhje për `class Node`:

1. Klasë e veçantë në `LinkedList.java`

```
public class LinkedList implements List {
  protected Node head;
  protected Node tail;
  protected int count;
  //...
}
class Node {
  Object element;
  Node next;
  Node previous;
}
```

Organizimi i fajlave. (Vazhdim)

2. Klasë e brendshme e `LinkedList.class` në `LinkedList.java`:

```
public class LinkedList implements List {
  protected Node head;
  protected Node tail;
  protected int count;
  protected static class Node {
    Object element;
    Node next;
    Node previous;
  }
  //...
}
```

Zbatimi i formës kanonike

Udhëzues disenji: Forma kanonike e klasave publike

Një klasë publike duhet të ndjekë *formën kanonike:*

- Të ofrojë konstruktor publik pa parametra
- Të mbikalojë metodën `equals()`
- Të mbikalojë metodën `toString()`
- Të implementojë `interface Cloneable`,
  dhe të mbikalojë metodën `clone()` nëse është e nevojshme
- Në qoftë se instanca të klasës do të ruhen në fajla
  ose të transferohen nëpër rrjet,
  të implementojë `interface java.io.Serializable`.

Ekuivalenca e objekteve

```
public boolean equals(Object other) {
  boolean result = false;
  if (other != null && other instanceof List) {
    List otherList = (List)other;
```

Ekuivalenca e objekteve. (Vazhdim)

```
    if (this.getCount() == otherList.getCount()) {
      boolean found = false;
      int i = 0;
      while (!found && i < this.getCount()) {
        Object element = this.get(i);
        Object otherElement = otherList.get(i);
        found = (element != null && !element.equals(otherElement))
                || (element == null && otherElement != null);
        i++;
      }
      result = !found;
    }
  }
  return result;
}
```

Klonimi

- Metoda `clone()` duhet të kthejë një klon të vetë objektit.
- Kloni nuk duhet të jetë i njëjti objekt;
  d.m.th., `o.clone() != o`
- Kloni duhet të jetë euivalent me objektin;
  d.m.th., `o.clone().equals(o)`

Klonimi. (Vazhdim)

```
public class Point implements Cloneable {
  protected double x, y;
  //deklarimet tjera
  public Object clone() throws CloneNotSupportedException {
    return super.clone();
  }
}
```

Klonimi. (Vazhdim)

```
Point p7 = new Point();
Point p8 = new Point();
try {
  p8 = (Point)p7.clone();
  System.out.println(p8.equals(p7));
}
catch (CloneNotSupportedException e) {
  e.printStackTrace();
}
```

Klonimi. (Vazhdim)

- Metoda `clone()` e klasës `Object` krijon një *kopje të cekët* të objektit, e jo një *kopje të thellë*.
- Për të mbështetur klonim të thellë të një objekti të gjithë anëtarët e objektit duhet të jenë të klonueshëm dhe metoda `clone()` duhet të jetë publike.

Klonimi. (Vazhdim)

```
protected static interface CloneableElement
      extends Cloneable {
  public Object clone() throws CloneNotSupportedException;
}
```

Klonimi. (Vazhdim)

```
public Object clone() throws CloneNotSupportedException {
  LinkedList result = new LinkedList();
  for (Node node = head; node != null; node = node.next) {
    Object e = null;
    if (node.element != null) {
      if (node.element instanceof CloneableElement) {
        e = ((CloneableElement)node.element).clone();
      }
      else {
        throw new CloneNotSupportedException();
      }
    }
    result.addTail(e);
  }
  return result;
}
```

Conversioni në String

Metoda toString()

Metoda toString() kthen paraqitjen si String të një objekti.

```
public String toString() {
  StringBuffer buff = new StringBuffer("{");
  int i = 0;
  for (Node node = head; node != null; node = node.next, i++) {
    if (i != 0) {
      buff.append(",");
    }
    buff.append(node.element);
  }
  buff.append("}");
  return buff.toString();
}
```

Organizimi i klasave

Udhëzues disenji: Përgjegjësia e metodave publike

- *Kompletësia:* Metodat publike duhet të ofrojnë qasje të plotë funksionalitetit të klasës
- *Siguria:* Invokimi i çfarëdo metode publike me çfarëdo renditje nuk guxon të rezultojë me gjendje jokonsistente.

Dokumentimi i kodit burimor

Dokumentimi me anë të `javadoc`

Dokumentacion i plotë referencimi i kodit burimor
mund të ndërtohet me anë të veglës `javadoc` të JDK:

1. Secilës klasë dhe secilës metodë publike i shtohet
   një grup komentesh të formatizuara në mënyrë të posaçme.
2. Ekzekutohet `javadoc` kundrejt pakos ose fajlit të klasës.

Dokumentimi i kodit burimor. (Vazhdim)

Tagu	Përshkrimi
`@author`	Autorët e kodit.
`@version`	Versioni aktual.
`@param`	Parametrat e një metode.
`@return`	Vlera e kthyer e një metode.
`@see`	Link nga dokumentacioni i tjera klasash ose metodash relevante.

Tabela: Disa tagje të shfrytëzuara shpesh

Testimi i njësive

Testimi i një klase

Një testim i tërësishëm i një klase do të duhej së paku që:

- të invokojë secilën metodë së paku njëherë;
- të invokojë secilën metodë me kombinime të ndryshme të vlerave të mundshme të parametrave;
- të ekzekutojë secilin urdhër dhe secilën rrugë të mundshme të secilës metodë së paku njëherë

Apleti i përmirësuar i orës digjitale

```
import java.awt.*;
public class DigitalClock2 extends DigitalClock {
  public void init() {
    String param = getParameter("color");
    if (param.equals("red")) {
      color = Color.red;
    } else if (param.equals("blue")) {
      color = Color.blue;
    } else if (param.equals("yelow")) {
      color = Color.yellow;
    } else if (param.equals("orange")) {
      color = Color.orange;
    } else {
      color = Color.green;
    }
  }
}
```

Apleti i përmirësuar i orës digjitale. (Vazhdim)

```
<applet code = "DigitalClock2.class"
  width=300 height=350>
  <param name=color value=red>
</applet>
```

Apleti i tekstit rrëshqitës

```
import javax.swing.*;
import java.awt.*;
public class ScrollingBanner
        extends JApplet implements Runnable {
  protected Thread bannerThread;
  protected String text;
  protected Font font =
    new java.awt.Font("Sans-serif", Font.BOLD, 24);
  protected int x, y;
  protected int delay = 50;
  protected int offset = 1;
  protected Dimension d;
```

Apleti i tekstit rrëshqitës. (Vazhdimi)

```
public void init() {
  String input = getParameter("delay");
  if (input != null) {
    delay = new Integer(input).intValue();
  }
  input = getParameter("text");
  if (input != null) {
    text = input;
  } else {
    text = "Tekst rrëshqitës!";
  }
  d = getSize();
  x = d.width;
  y = font.getSize();
}
```

Apleti i tekstit rrëshqitës. (Vazhdimi)

```
public void paint(Graphics g) {
  g.setFont(font);
  FontMetrics fm = g.getFontMetrics();
  int length = fm.stringWidth(text);
  g.setColor(Color.black);
  g.drawString(text, x, y);
  x -= offset;
  if (x < -length) {
    x = d.width;
  }
  g.setColor(Color.green);
  g.drawString(text, x, y);
}
```

Apleti i tekstit rrëshqitës. (Vazhdimi)

```
public void start() {
  if (bannerThread == null) {
    bannerThread = new Thread(this);
    bannerThread.start();
  }
}

public void stop() {
  bannerThread = null;
}
```

Apleti i tekstit rrëshqitës. (Vazhdimi)

```
  public void run() {
    while (Thread.currentThread() == bannerThread) {
      repaint();
      try {
        Thread.sleep(delay);
      }
      catch (InterruptedException e) {}
    }
  }
}
```

Apleti i tekstit rrëshqitës. (Vazhdimi)

```
<applet code = "ScrollingBanner.class"
  width=300 height=350>
  <param name=delay value=10>
  <param name=text value="Java është cool">
</applet>
```

Idiomë: Aplet animues

```
public class AnimationApplet
        extends JApplet implements Runnable {
  private Thread mainThread;
  private int delay;

  public void start() {
    if (mainThread == null) {
      mainThread = new Thread(this);
      mainThread.start();
    }
  }

  public void stop() {
    mainThread = null;
  }
```

Idiomë: Aplet animues. (Vazhdim)

```
  public void run() {
    while (Thread.currentThread() == mainThread) {
      repaint();
      try {
        Thread.sleep(delay);
      }
      catch (InterruptedException e) {}
    }
  }

  public void paint(Graphics g) {
    //ngjyrosja e kornizës aktuale
  }
  //...
}
```

Shfrytëzimi i baferimit të dyfishtë

- Versioni vijues shkakton „dridhje“ të imazhit.

```
public void paint(Graphics g) {
  g.setFont(font);
  FontMetrics fm = g.getFontMetrics();
  int length = fm.stringWidth(text);
  x -= offset;
  if (x < -length) {
    x = d.width;
  }
  g.setColor(Color.black);
  g.fillRect(0,0,d.width,d.height);
  g.setColor(Color.green);
  g.drawString(text, x, y);
}
```

Shfrytëzimi i baferimit të dyfishtë. (Vazhdimi)

```
import java.awt.*;
public class ScrollingBanner2 extends ScrollingBanner {
  protected Image image;
  protected Graphics offscreen;
  public void update(Graphics g) {
    if (image == null) {
      image = createImage(d.width, d.height);
      offscreen = image.getGraphics();
    }
    super.paint(offscreen);
    g.drawImage(image, 0, 0, this);
  }
```

Shfrytëzimi i baferimit të dyfishtë. (Vazhdimi)

```
  public void paint(Graphics g) {
    update(g);
  }
}
```

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

- Mbingarkimi i metodave dhe konstruktorëve
- Trashigimia
  - Zgjerimi i klasave
  - Implementimi dhe zgjerimi i interfejsave
- Relacioni nëntip
- Mbikalimi i metodave
  - Polimorfizmi