Struktura e kontrollit: Urdhërat kushtëzues

Struktura e kontrollit: Urdhërat kushtëzues

Objektivat:

- Të studiohet rrjedha e kontrollit të programit dhe një formë e re urdhëri, urdhëri kushtëzues, e cila i mundëson metodës të bëjë pyetje mbi vlerat e argumenteve të saj ose të dhënave hyrëse. Mbështetur në përgjegjjet në pyetjet, metoda mund të kontrollojë rrjedhën e llogaritjes në rrugën më të përshtatshme për këto vlera.
- Të studiohen operatorët logjikë dhe të mësohet logjika prapa urdhërave kushtëzues.
- Të shfrytëzohet dituria mbi rrjedhën e kontrollit për të zhvilluar ndërtim aplikacionesh në më tepër detaje; në veçanti, aplikacioni ndahet në tri komponente: model, view (pamje) dhe controller (kontrollues), ku view merr përgjegjësinë për hyrjen dhe daljen e të dhënave, kontrolluesi për rrjedhën e kontrollit, kurse modeli merr përgjegjësinë për llogaritjet e përgjegjjeve.

Rrjedha e kontrollit dhe struktura e kontrollit

Rrjedhë e kontrollit: renditja në të cilën ekzekutohen urdhërat e një programi.

Strukturë e kontrollit: formë urdhëri e cila përcakton rrjedhën e kontrollit.

Rrjedha e kontrollit dhe struktura e kontrollit – Vazhdim

Struktura e kontrollit e vargut të urdhërave

```
URDHËRI1;
URDHËRI2;
...
URDHËRIn;
```

përcakton rrjedhën e kontrollit të ekzekutimit të njëpasnjëshëm të urdhërave përkatëse.

Struktura e kontrollit e invokimit të metodës

```
PRANUESI.METODA(ARGUMENTET)
```

përcakton rrjedhën e kontrollit të pauzimit të ekzekutimit në vendin e invokimit ashtu që urdhërat e emërtuara me *METODA* në *PRANUESI* të ekzekutohen në vijim.

Struktura kushtëzuese e kontrollit

Urdhër kushtëzues („*urdhër* `if`“): strukturë kontrolli e cila shtron një pyetje (*test*) dhe në bazë të përgjegjjes zgjedh një rrjedhë të kontrollit.

Sintaksa e urdhërit kushtëzues:

```
if ( TESTI )
{ URDHËRAT1 }
else { URDHËRAT2 }
```

Semantika: në qoftë se rezultati i testit është `true`, atëherë ekzekutohet *then-krahu*; në qoftë se rezultati i testit është `false`, atëherë ekzekutohet *else-krahu*.

Struktura kushtëzuese e kontrollit – Vazhdim

```
/** Afishon se a është argumenti negativ.
 *  @param i - argumenti */
public void printPolarity(int i)
{ System.out.print("Numri " + i + " është ");
  if ( i < 0 )
  { System.out.println("negativ.");}
  else { System.out.println("jonegativ.");}
  System.out.println("Fund!");
}
```

Struktura kushtëzuese e kontrollit – Vazhdim

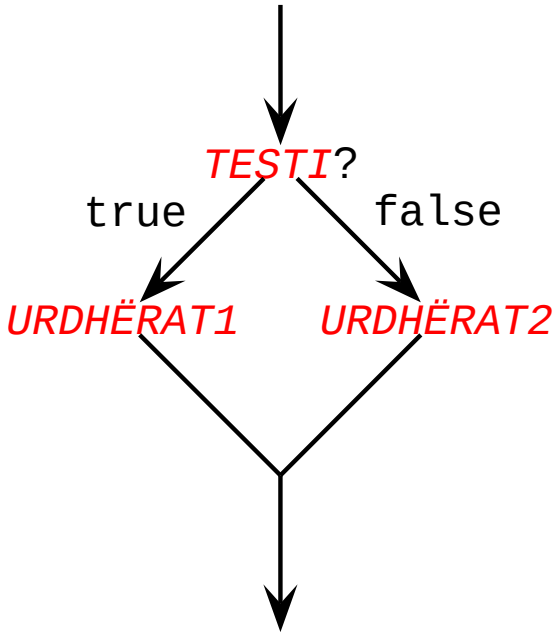

Figura 1. Skema e rrjedhës së kontrollit të përcaktuar nga urdhëri kushtëzues

Struktura kushtëzuese e kontrollit – Vazhdim

Ndonjëherë, kur dëshirohet të specifikohet vetëm then-krahu, urdhëri kushtëzues shkruhet në formën

```
if ( TESTI )
{ URDHËRAT }
```

e cila është shkurtesë për

```
if ( TESTI ) { URDHËRAT } else { }
```

Struktura kushtëzuese e kontrollit – Vazhdim

Algoritmi i një aplikacioni i cili konverton orët në sekonda:

1. Lexo nga shfrytëzuesi (si hyrje) një numër të plotë, i cili paraqet sasinë e orëve.
2. Në qoftë se numri i plotë është jonegativ, atëherë kthe orët në sekonda. Përndryshe, numri i plotë nuk është jonegativ, prandaj afisho një mesazh gabimi.

Struktura kushtëzuese e kontrollit – Vazhdim

```
import javax.swing.*;
/** Konverton orët në sekonda.
 * Input: numër i plotë jonegativ
 * Output: koha e konvertuar në sekonda */
public class ConvertHours
{ public static void main(String[] args)
  { int hours = new Integer(
      JOptionPane.showInputDialog("Numri i orëve")).intValue();
```

Struktura kushtëzuese e kontrollit – Vazhdim

```
    if ( hours >= 0 )
    { int seconds = hours * 60 * 60;
      JOptionPane.showMessageDialog(null,
        hours + " orë janë " + seconds + " sekonda");
    }
    else
    { JOptionPane.showMessageDialog(null,
        "Gabim: hyrje negative " + hours);
    }
  }
}
```

Urdhërat kushtëzues të vendosur përbrenda

Urdhëra kushtëzues mund të vendosen përbrenda ndonjërit nga krahët e një urdhëri kushtëzues tjetër, për të koduar situata kur ekzistojnë pyetje të shumëfishta për t'u testuar.

Urdhërat kushtëzues të vendosur përbrenda – Vazhdim

Algoritmi për një version më të avansuar të `MakeChange`:

1. Lexo eurot si një numër të plotë.
2. Në qoftë se eurot janë negativë, atëherë gjenero një mesazh gabimi. Përndryshe, eurot janë jonegativë:
   (a) Lexo centët, numër i plotë.
   (b) Në qoftë se centët janë negativë, atëherë gjenero një mesazh gabimi. Përndryshe, centët janë jonegativë:
      i. Në qoftë se centët janë më të mëdhenj se 99, atëherë gjenero një mesazh gabimi. Përndryshe, centët janë në intrvalin 0. . . 99:
         A. Llogarit totalin e eurove dhe centëve.
         B. Nxjerr sasinë përkatëse të metelikëve gjysëmeurosh, njëzetqindarkësh, dhjetëqindarkësh, pesëqindarkësh dhe qindarkësh.

Urdhërat kushtëzues të vendosur përbrenda – Vazhdim

```
import javax.swing.*;
/** Llogarit sasinë e metelikeve.
 *  Input: dy numra të dhënë në rreshtin komandues
 *         sasia e eurove dhe sasia e centëve
 *  Output: sasitë e metelikëve */
public class MakeChange
{ public static void main (String[] args)
  { int euros = new Integer(args[0]).intValue();
    if ( euros < 0 )
    { JOptionPane.showMessageDialog(null,
        "Gabim: euro negativë " + euros);
    }
```

Urdhërat kushtëzues të vendosur përbrenda – Vazhdim

```
else
{ int cents = new Integer(args[1]).intValue();
  if ( cents < 0 )
  { JOptionPane.showMessageDialog(null,
      "Gabim: cent negativë " + cents);
  }
  else
  { if ( cents > 99 )
    { JOptionPane.showMessageDialog(null,
        "Gabim: në cent " + cents);
    }
```

Urdhërat kushtëzues të vendosur përbrenda – Vazhdim

```
        else
        { int money = (euros * 100) + cents;
          System.out.println("Gjysma = " + (money / 50));
          money = money % 50;
          System.out.println("Njëzetqindarka = " + (money / 20));
          money = money % 20;
          System.out.println("Dhjetëqindarka = " + (money / 10));
          money = money % 10;
          System.out.println("Pesëqindarka = " + (money / 5));
          money = money % 5;
          System.out.println("Qindarka = " + money);
        }
      }
    }
  }
}
```

Urdhërat kushtëzues të vendosur përbrenda – Vazhdim

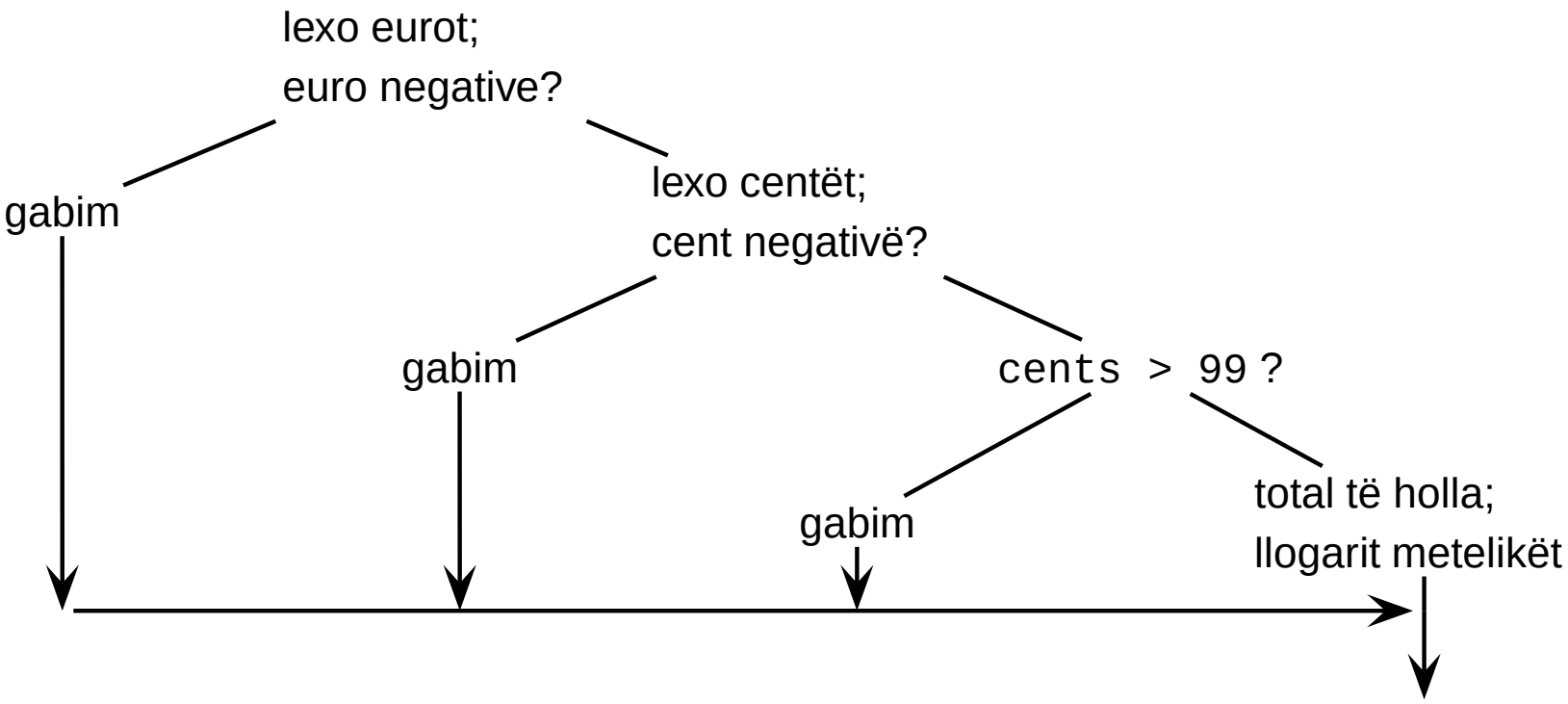

Figura 2. Skema e rrjedhës së kontrollit

Urdhërat kushtëzues të vendosur përbrenda – Vazhdim

```
import javax.swing.*;
/** Llogarit sasinë e metelikeve
 *  Input: dy numra të dhënë në rreshtin komandues
 *         sasia e eurove dhe sasia e centëve
 *  Output: sasitë e metelikëve */
public class MakeChange4
{ public static void main (String[] args)
  { boolean ok = true;  // memoron korrektësinë e të dhënave hyrëse
    int euros = new Integer(args[0]).intValue();
    int cents = new Integer(args[1]).intValue();
    if ( euros < 0 )
    { JOptionPane.showMessageDialog(null,
        "Gabim: euro negativë " + euros);
      ok = false;   // gabimi mohon korrektësinë
    }
```

Urdhërat kushtëzues të vendosur përbrenda – Vazhdim

```
if ( ok )   // eurot janë në rregull, shqyrto centët
{ if ( cents < 0 )
  { JOptionPane.showMessageDialog(null,
      "Gabim: cent negativë " + cents);
    ok = false;
  }
  if ( cents > 99 )
  { JOptionPane.showMessageDialog(null,
      "Gabim: në cent " + cents);
    ok = false;
  }
}
```

Urdhërat kushtëzues të vendosur përbrenda – Vazhdim

```
    if ( ok )   // eurot dhe centët janë në rregull,
                //   llogarit përgjegjjen.
    { int money = (euros * 100) + cents;
      System.out.println("Gjysma = " + (money / 50));
      money = money % 50;
      System.out.println("Njëzetqindarka = " + (money / 20));
      money = money % 20;
      System.out.println("Dhjetëqindarka = " + (money / 10));
      money = money % 10;
      System.out.println("Pesëqindarka = " + (money / 5));
      money = money % 5;
      System.out.println("Qindarka = " + money);
    }
  }
}
```

Operatorët logjikë

```
import javax.swing.*;
/** Llogarit sasinë e metelikeve
 *  Input: dy numra të dhënë në rreshtin komandues
 *         sasia e eurove dhe sasia e centëve
 *  Output: sasitë e metelikëve */
public class MakeChange
{ public static void main (String[] args)
  { int euros = new Integer(args[0]).intValue();
    int cents = new Integer(args[1]).intValue();
    if ( (euros < 0) || (cents < 0) || (cents > 99) )
    { JOptionPane.showMessageDialog(null,
        "Gabim në hyrje " + euros + " " + cents); }
```

Operatorët logjikë – Vazhdim

```
    else
    { int money = (euros * 100) + cents;
      System.out.println("Gjysma = " + (money / 50));
      money = money % 50;
      System.out.println("Njëzetqindarka = " + (money / 20));
      money = money % 20;
      System.out.println("Dhjetëqindarka = " + (money / 10));
      money = money % 10;
      System.out.println("Pesëqindarka = " + (money / 5));
      money = money % 5;
      System.out.println("Qindarka = " + money);
    }
  }
}
```

Operatorët logjikë – Vazhdim

Operator logjik: operator i cili mundëson të shkruhen dy krahasime në një shprehje të tipit `boolean`.

Operatori	Semantika

SH1 && *SH2*	Konjuksioni „dhe“:
	`true && true => true`
	`true && false => false`
	`false && SH2 => false`

SH1 || *SH2*	Disjunksioni „ose“:
	`false || false => false`
	`false || true => true`
	`true || SH2 => true`

!*SH*	Negacioni „jo“:
	`!true => false`
	`!false => true`

Tabela 1. Operatorët logjikë

Operatorët logjikë – Vazhdim

Semantika e urdhërave logjikë:

```
(minutes >= 0) && (minutes < 60)
=> false && (minutes < 60)
=> false

(cents < 0) || (cents > 99)
=> true || (cents > 99)
=> true
```

Operatorët logjikë – Vazhdim

Operatorët logjikë kanë precedencë më të ulët sesa operatorët relacionalë; p.sh., në versionin e fundit të aplikacionit `MakeChange` kemi mundur të shkruajmë

```
if ( euros < 0 || cents < 0 || cents > 99 )
```

Zbatimi i urdhërave kushtëzues

Dy zbatimet kryesore të urdhërave kushtëzues janë:

- për t'i mundësuar metodës të veprojë bazuar në vlera të dhënash dhe parametrash;
- për të paisur metodën për mbrojtje nga të dhëna dhe parametra aktualë të pakuptimë.

Zbatimi i urdhërave kushtëzues – Vazhdim

```
/** Konverton kohën 12 orëshe në 24 orëshe.
 *  @param hour - koha 12 orëshe në orë
 *  @param minute - koha në minuta
 *  @param am - boolean: true për mëngjes, false për pasdite
 *  @return koha 24 orëshe, e formatizuar si string
 */
public String twentyfourHourClock(int hour, int minute, boolean am)
{ String answer = "";
  if ( hour < 1 || hour > 12 || minute < 0 || minute > 59 )
  { answer = "Gabim: " + hour + ":" + minute; }
```

Zbatimi i urdhërave kushtëzues – Vazhdim

```
  else { if ( hour == 12 )
         { if ( am )
           { answer = answer + "0"; }
           else { answer = answer + "12"; }
         }
         else { if ( am )
                { answer = answer + hour; }
                else { answer = answer + (hour + 12); }
              }
         answer = answer + ":";
         if ( minute < 10 )
         { answer = answer + "0"; }
         answer = answer + minute;
       }
  return answer;
}
```

Ndryshimi i rrjedhës së kontrollit

Përjashtim (*exception*): gabim i cili ndodh gjatë ekzekutimit të një programi. Zakonisht, përjashtimi ndalon programin (themi se është *hedhur* një përjashtim).

Hedhja eksplicite e një përjashtimi mund të bëhet me anë të strukturës kontrolluese:

```
throw new RuntimeException(MESAZHI);
```

ku *MESAZHI* është një string.

Ndryshimi i rrjedhës së kontrollit – Vazhdim

```
/** hourToSecond konverton orët në sekonda
 *  @param hours - orët
 *  @return ekuvalenti në sekonda */
public int hourToSecond(int hours)
{ int seconds = 0;
  if ( hours < 0 )
  { String error = "Gabim: orët " + hours;
    JOptionPane.showMessageDialog(null, error);
    throw new RuntimeException(error);
  }
  else { seconds = hours * 60 * 60; }
  return seconds;
}
```

Ndryshimi i rrjedhës së kontrollit – Vazhdim

Urdhëri `return` shkakton që rrjedha e kontrollit të kthehet menjëherë në vendin e invokimit të metodës.

```
public void printSeconds(int hours)
{ if ( hours < 0 )
  { return; }
  int seconds = hours * 60 * 60;
  System.out.println(horus + " orë janë " + seconds + " sekonda");
}

public int hourToSecond(int hours)
{ if ( hours < 0 )
  { return -1; }
  int seconds = hours * 60 * 60;
  return seconds;
}
```

Urdhëri `switch`

Në disa raste, urdhëri `switch` është një alternativë për urdhërat if-else të vendosur përbrenda.

Sintaksa:

```
switch (SHPREHJA)
{ case VLERA1 : { URDHËRAT1;
                  break;
                }
  case VLERA2 : { URDHËRAT2;
                  break;
                }
  ...
  case VLERAn : { URDHËRATn;
                  break;
                }
  default : { URDHËRATn+1; }
}
```

Urdhëri `switch` – Vazhdim

Semantika:

1. Llogaritet *SHPREHJA* në vlerën e saj, të themi, v.
2. Në qoftë se ndonjë nga *VLERAk* është e barabartë me v, atëherë ekzekutohen urdhërat përkatëse *URDHËRATk*; në qoftë se asnjë nga *VLERAk* nuk është e barabartë me v, atëherë ekzekutohen urdhërat *URDHËRATn+1*;

Urdhëri switch – Vazhdim

Shembull:

```
switch(ch)
{ case ’C’: case ’c’: { // Celsius
    // ...
    break;
  }
  case ’F’: case ’f’: { // Fahrenheit
    // ...
    break;
  }
  case ’K’: case ’k’: { // Kelvin
    // ...
    break;
  }
  default: { System.out.println("Gabim: TempConverter"); }
}
```

Komponentet model dhe kontrollues

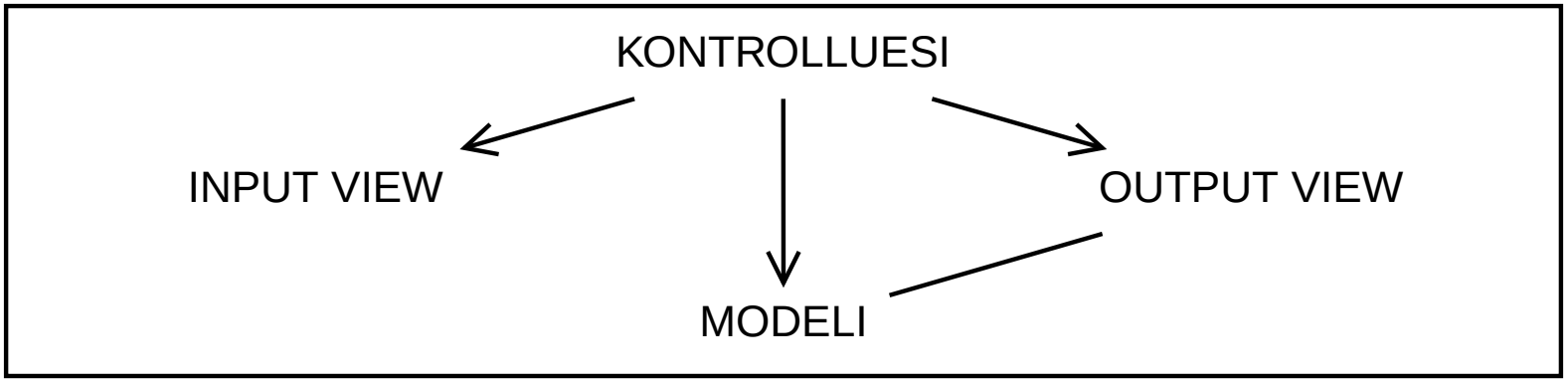

Figura 3. Një arkitekturë e thjeshtë MVC

Komponenta model merr përgjegjësinë për „modelimin“ e problemit dhe „kompjutimin“ e pergjegjjeve të tij.

Komponenta view merr përgjegjësinë për bartjen e të dhënave hyrëse dhe dalëse.

Komponenta kontrollues merr përgjegjësinë për rrjedhën e kontrollit të aplikacionit, d.m.th. përcakton renditjen e kryerjes së kompjutimit.

Komponentet model dhe kontrollues – Vazhdim

Arsyet kryesore për zbatimin e MVC arkitekturës:

1. Klasët të cilat i gjenerojnë objektet mund të rishfrytëzohen.
2. Komponentet e organizojnë një aplikacion në pjesë madhësish të menagjueshme me detyra specifike. Pjesët janë të organizuara në një arkitekturë standarde dhe mund të shkruhen dhe testohen veç e veç.
3. Komponentet janë të izoluara, kështu që modifikimi i njërës pjesë nuk kërkon rishkrimin edhe të pjesëve tjera.

Komponentet model dhe kontrollues – Vazhdim

Në një situatë ideale, ndërtimi i një aplikacioni do të konsistonte nga

- zgjedhja e klasëve tanimë të shkruara, ndoshta paksa të zgjeruara, për gjenerimin e input-output-views;
- zgjedhja një klase tanimë të shkruar, ndoshta paksa të zgjeruar, për gjenerimin e modelit;
- shkruarja e një klase të re për gjenerimin e kontrolluesit i cili lidh së bashku pjesët e aplikacionit.

Komponentet model dhe kontrollues – Vazhdim

Model: komponentë në aplikacion e cila „modelon“ problemin që programohet dhe llogarit përgjegjjet e tij.

Atribut: variabël fushe private përbrenda një klase modeli. Fusha përmban informata mbi „gjendjen e brendshme“ të modelit.

Case study: menagjer kontosh bankiere

Ndërtojmë një menagjer kontoje bankiere i cili për hyrjet, p.sh.,

```
Komanda (D,T,M): d 50.50
Komanda (D,T,M): T 30.33
Komanda (D,T,M): M
```

afishon në dritare grafike raportet mbi transakcionet dhe gjendjet e balancit.

Disenjimi i modelit

`BankAccount` modelon një konto bankiere	
Konstruktori:	
`BankAccount(int amount)`	inicializon konton
Metodat:	
`boolean deposit(int amount)`	deponon sasinë e dhënë në konton; kthen `true` për sukses dhe `false` për mossukses
`boolean withdraw(int amount)`	tërheq sasinë e dhënë nga kontoja; kthen `true` për sukses dhe `false` për mossukses
`int getBalance()`	kthen balansin e kontos

Tabela 2. Interfejsi i klasës `BankAccount`

Disenjimi i modelit – Vazhdim

```
import javax.swing.*;
/** Modelon një konto. */
public class BankAccount
{ private int balance;  // invariantë: balance >= 0
  /** Konstruktori.
   *  @param amount - balansi inicial */
  public BankAccount(int amount)
  { if ( amount >= 0 )
    { balance = amount; }
    else { balance = 0; }
  }
```

Disenjimi i modelit – Vazhdim

```
/** Deponon të holla në konto.
 *  @param amount - sasia për deponim
 *  @return true nëse tërheqja e suksesshme, false përndryshe */
public boolean deposit(int amount)
{ boolean result = false;
  if ( amount < 0 )
  { JOptionPane.showMessageDialog(null,
      "Gabim: depositi " + amount);
  }
  else { balance = balance + amount;
         result = true;
       }
  return result;
}
```

Disenjimi i modelit – Vazhdim

```
/** Tërheq të holla nga kontoja.
 *  @param amount - sasia për tërheqje
 *  @return true nëse tërheqja e suksesshme, false përndryshe */
public boolean withdraw(int amount)
{ boolean result = false;
  if ( amount < 0 )
  { JOptionPane.showMessageDialog(null,
      "Gabim: tërheqja " + amount);
  }
  else if ( amount > balance )
       { JOptionPane.showMessageDialog(null,
           "Gabim: tërheqja tejkalon balansin");
       }
  else { balance = balance - amount;
           result = true;
       }
  return result;
}
```

Disenjimi i modelit – Vazhdim

```
  /** getBalance kthen balansin vijues
   *  @return balansi */
  public int getBalance()
  { return balance; }
}
```

Disenjimi i modelit – Vazhdim

Metodë accessor: metodë e cila vetëm raporton vlera fushash të objektit (atributesh) por nuk ndyshon asnjë fushë.

Metodë mutator: metodë e cila ndyshon vlera fushash të një objekti.

Arkitektura e softuerit të aplikacionit

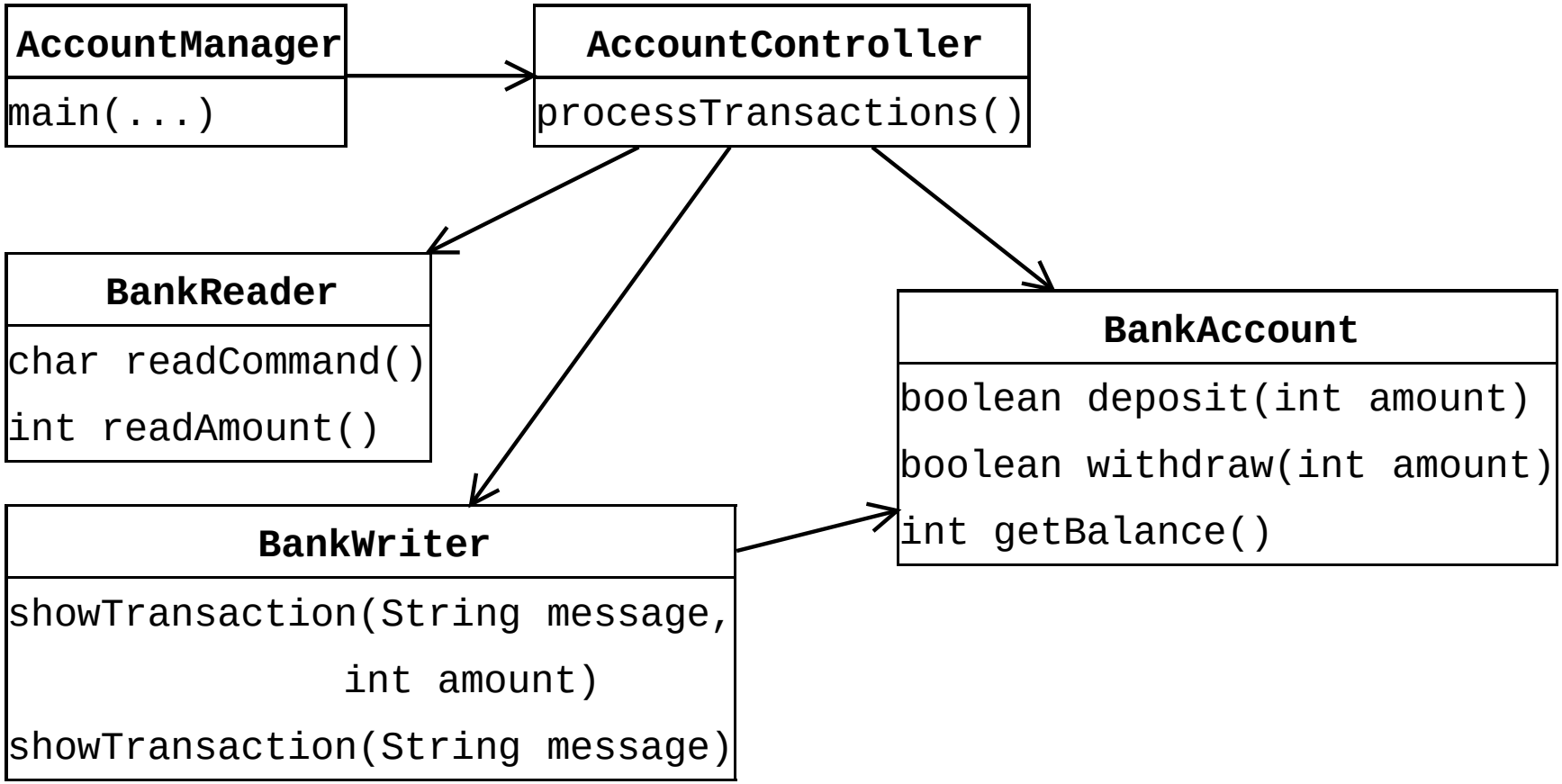

Figura 4. Arkitektura e `AccountManager`

View komponentet

`BankReader` lexon transakcionet nga shfry-tëzuesi	
Metodat:	
`char readCommand(String message)`	lexon linjë të re komanduese; kthen karakterin e parë të linjës
`int readAmount()`	kthen vlerën numerike nga linja e lexuar së fundi

Tabela 3. Interfejsi i klasës `BankReader`

View komponentet – Vazhdim

`BankWriter` shkruan transakcionet	
Konstruktori:	
`BankWriter(String title, BankAccount a)`	inicializon shkruesin me titullin dhe konton e dhënë
Metodat:	
`showTransaction(String message, int amount)`	afishon transakcionin e dhënë me vlerën e dhënë
`showTransaction(String message)`	afishon transakcionin e dhënë

Tabela 4. Interfejsi i klasës `BankWriter`

View komponentet – Vazhdim

```
import javax.swing.*;
/** Lexon transakcione bankiere. */
public class BankReader
{ private String inputLine;
  /** Konstruktori. */
  public BankReader()
  { inputLine = ""; }
  /** Lexon linjë të re komanduese.
   *  @param message - prompti për shfrytëzuesin
   *  @return karakterin e parë të komandës */
  public char readCommand(String message)
  { inputLine
      = JOptionPane.showInputDialog(message).trim().toUpperCase();
    return inputLine.charAt(0);
  }
```

View komponentet – Vazhdim

```
  /** Kthen vlerën numerike nga linja komanduese.
   *  @return vlera e konvertuar në cent */
  public int readAmount()
  { int answer = 0;
    String s = inputLine.substring(1, inputLine.length());
    s = s.trim();
    if ( s.length() > 0)
    { double money = new Double(s).doubleValue();
      answer = (int)(money * 100);
    }
    else { JOptionPane.showMessageDialog(null,
              "Gabim: transakcion pa vlerë");
         }
    return answer;
  }
}
```

View komponentet – Vazhdim

```
import javax.swing.*;
import java.awt.*;
import java.text.*;
/** Shkruan transakcione bankiere. */
public class BankWriter extends JPanel
{
  private int width = 300;
  private int height = 200;
  private BankAccount account;
  private String lastTransaction = "";
```

View komponentet – Vazhdim

```
/** Konstruktori.
 *  @param title - titulli i dritares grafike
 *  @param a - kontoja bankiere që afishohet */
public BankWriter(String title, BankAccount a)
{ account = a;
  JFrame f = new JFrame();
  f.getContentPane().add(this);
  f.setTitle(title);
  f.setSize(width, height);
  f.setBackground(Color.white);
  f.setVisible(true);
}
```

View komponentet – Vazhdim

```
public void paintComponent(Graphics g)
{ g.setColor(Color.white);
  g.fillRect(0, 0, width, height);
  g.setColor(Color.black);
  int textMargin = 50;
  int textBaseline = 50;
  g.drawString(lastTransaction, textMargin, textBaseline);
  g.drawString("Balanci vijues = "
      + unconvert(account.getBalance())
      + " E", textMargin, textBaseline + 20);
}
/** Reformaton centët në euro dhe cent. */
private String unconvert(int amount)
{ double money = amount / 100.0;
  return new DecimalFormat("0.00").format(money);
}
```

View komponentet – Vazhdim

```
  /** Afishon transakcion.
   *  @param message - transakcioni
   *  @param amount - sasia e transakcionit */
  public void showTransaction(String message, int amount)
  { lastTransaction = message + " " + unconvert(amount) + " E";
    this.repaint();
  }
  /** Afishon transakcion.
   *  @param message - transakcioni */
  public void showTransaction(String message)
  { lastTransaction = message;
    this.repaint();
  }
}
```

Konstruktimi i kontrollerit

Algoritmi i kontrollerit për procesimin e një transakcioni:

1. Lexo komandën.
2. Në qoftë se komanda është `'M'`, atëherë përfundo procesimin.
3. Përndryshe:
   (a) Në rast se komanda është `'D'`, lexo sasinë e depositit, kryej depositin dhe afisho transakcionin.
   (b) Në rast se komanda është `'T'`, lexo sasinë e tërheqjes, kryej tërheqjen dhe afisho transakcionin.
   (c) Përndryshe, komanda është jolegale, prandaj ankohu.

Konstruktimi i kontrollerit – Vazhdim

```
/** Menagjon një konto.
 *  Input: një varg komandash të formës
 *    D ee.cc (deponim), T ee.cc (tërheqje), M (mjaft)
 *  Output: Afishon transakcione */
public class AccountController
{ private BankReader reader;
  private BankWriter writer;
  private BankAccount account;
  /** Konstruktori.
   *  @param r - input view objekti
   *  @param w - output view objekti
   *  @param a - model objekti */
  public AccountController(BankReader r, BankWriter w,
                           BankAccount a)
  { reader = r;
    writer = w;
    account = a;
  }
```

Konstruktimi i kontrollerit – Vazhdim

```
/** Proceson transakcione. */
public void processTransactions()
{ char command =
    reader.readCommand("Komanda (D,T,M) dhe sasia:");
  if ( command == 'M' )
  { }
  else
  { switch ( command )
    { case 'D' :
      { int amount = reader.readAmount();
        boolean ok = account.deposit(amount);
        if ( ok )
        { writer.showTransaction("Deponim", amount); }
        else { writer.showTransaction("Deponim jovalid"); }
        break;
      }
```

Konstruktimi i kontrollerit – Vazhdim

```
        case 'T' :
        { int amount = reader.readAmount();
          boolean ok = account.withdraw(amount);
          if ( ok )
          { writer.showTransaction("Tërheqje", amount); }
          else { writer.showTransaction("Tërheqje jovalide"); }
          break;
        }
        default :
        { writer.showTransaction("Komandë jovalide " + command); }
      }
      this.processTransactions();
    }
  }
}
```

Konstruktimi i kontrollerit – Vazhdim

```
/** Starton aplikacionin për menagjim kontoje. */
public class AccountManager
{
  public static void main(String[] args)
  { BankReader reader = new BankReader();
    BankAccount account = new BankAccount(0);
    BankWriter writer =
      new BankWriter("Shkruesi bankier", account);
    AccountController controller =
      new AccountController(reader, writer, account);
    controller.processTransactions();
  }
}
```

Konstruktimi i kontrollerit – Vazhdim

Invokim rekursiv: invokim ku metoda dërgon mesazh për të ristartuar vetvehten.

Emër i mbingarkuar metode (*overloaded*): emër metode i cili përdoret për të emërtuar dy metoda të ndryshme të së njëjtës klasë. Objekti pranues i mesazhit shfrytëzon sasinë dhe tipin e të dhënave të parametrave aktualë për të zgjedhur metodën përkatëse për ekzekutim.

Objekte të shumëfishta nga e njëjta klasë

Modifikojmë menagjerin e kontove ashtu që të menagjojë dy konto të ndryshme në mënyrë simultane.

Hyrjet, p.sh.:

```
Komanda (P,S,D,T,M): P
Komanda (P,S,D,T,M): d 50.50
Komanda (P,S,D,T,M): s
Komanda (P,S,D,T,M): D10.10
Komanda (P,S,D,T,M): p
Komanda (P,S,D,T,M): T 30.33
Komanda (P,S,D,T,M): m
```

Objekte të shumëfishta nga e njëjta klasë – Vazhdim

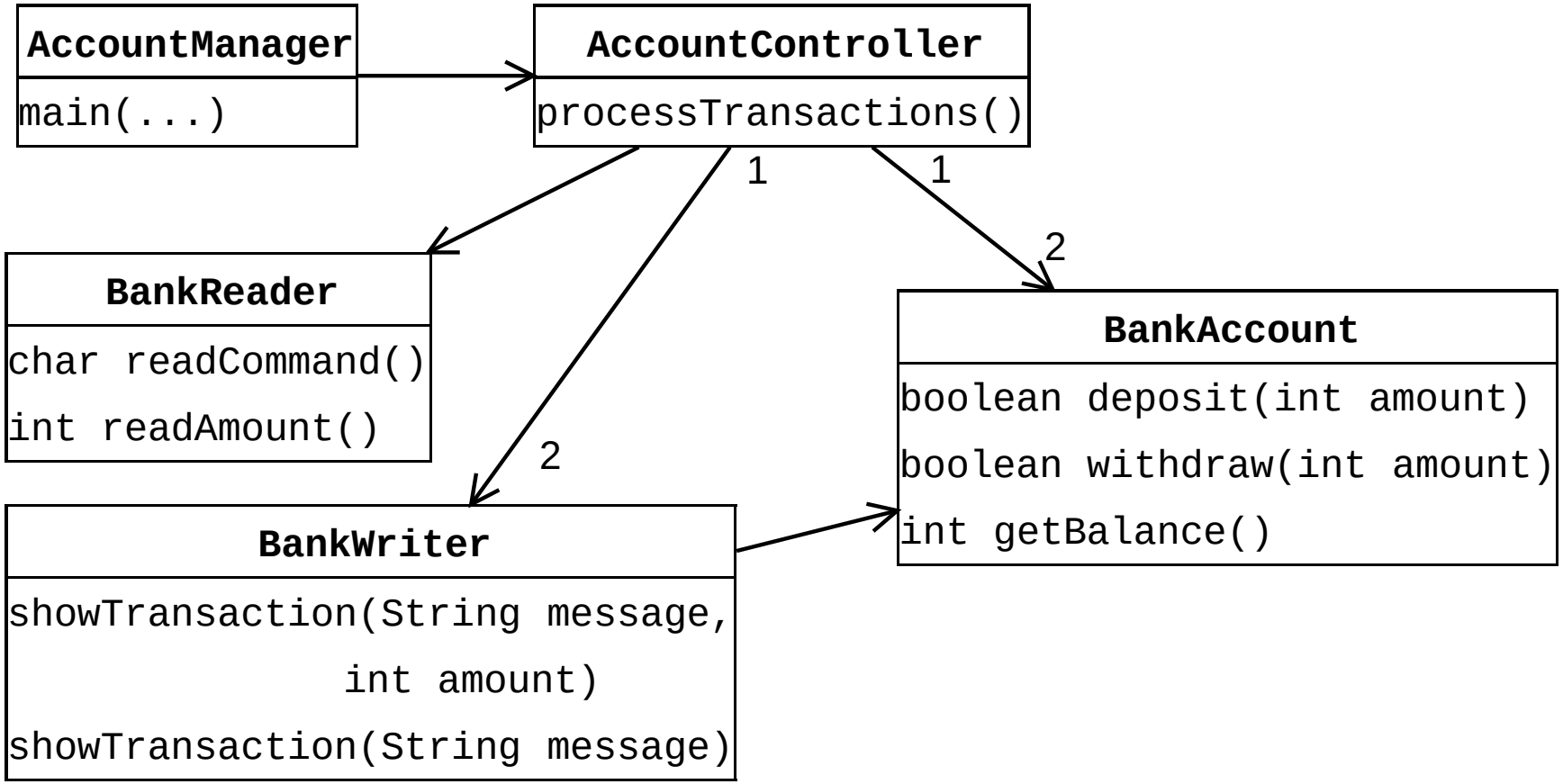

Figura 5. Arkitektura e `AccountManager`

Objekte të shumëfishta nga e njëjta klasë – Vazhdim

```
/** Menagjon dy konto.
 *  Input: një varg komandash të formës
 *    P (menagjo konton primare), S (menagjo konton sekondare)
 *    D ee.cc (deponim), T ee.cc (tërheqje), M (mjaft)
 *  Output: Afishon transakcione */
public class AccountController2
{ private BankReader reader;
  // kontoja primare
  private BankWriter primaryWriter;
  private BankAccount primaryAccount;
  // kontoja sekondare
  private BankWriter secondaryWriter;
  private BankAccount secondaryAccount;
  // kontoja aktive
  private BankWriter writer;
  private BankAccount account;
  // invariantë: barazi me fushat të kontos primare ose sekondare
```

Objekte të shumëfishta nga e njëjta klasë – Vazhdim

```
/** Konstruktori
 *  @param r - input view objekti
 *  @param w1 - output view objekti i parë
 *  @param a1 - model objekti i parë
 *  @param w2 - output view objekti i dytë
 *  @param a2 - model objekti i dytë */
public AccountController2(BankReader r,
                          BankWriter w1, BankAccount a1,
                          BankWriter w2, BankAccount a2)
{ reader = r;
  primaryWriter = w1;
  primaryAccount = a1;
  secondaryWriter = w2;
  secondaryAccount = a2;
  writer = primaryWriter;
  account = primaryAccount;
}
```

Objekte të shumëfishta nga e njëjta klasë – Vazhdim

```
/** Ndryshon konton aktive. */
private void resetAccount(BankWriter w, BankAccount a)
{ writer.showTransaction("Joaktiv");
  writer = w;
  account = a;
  writer.showTransaction("Aktiv");
}
```

Objekte të shumëfishta nga e njëjta klasë – Vazhdim

```
/** Proceson transakcione. */
public void processTransactions()
{ char command
    = reader.readCommand("Komanda (P,S,D,T,M) dhe sasia:");
  if ( command == 'M' )
  { }
  else
  { switch ( command )
    { case 'D' :
      { int amount = reader.readAmount();
        boolean ok = account.deposit(amount);
        if ( ok )
        { writer.showTransaction("Deponim", amount); }
        else { writer.showTransaction("Deponim jovalid"); }
        break;
      }
```

Objekte të shumëfishta nga e njëjta klasë – Vazhdim

```
case 'T' :
{ int amount = reader.readAmount();
  boolean ok = account.withdraw(amount);
  if ( ok )
  { writer.showTransaction("Tërheqje", amount); }
  else { writer.showTransaction("Tërheqje jovalide"); }
  break;
}
```

Objekte të shumëfishta nga e njëjta klasë – Vazhdim

```
        case 'P' :
        { this.resetAccount(primaryWriter, primaryAccount);
          break;
        }
        case 'S' :
        { this.resetAccount(secondaryWriter, secondaryAccount);
          break;
        }
        default :
        { writer.showTransaction("Komandë jovalide " + command); }
      }
      this.processTransactions();
    }
  }
}
```

Objekte të shumëfishta nga e njëjta klasë – Vazhdim

```
/** Starton aplikacionin për menagjim kontosh. */
public class AccountManager2
{
  public static void main(String[] args)
  { BankReader reader = new BankReader();
    BankAccount primaryAccount = new BankAccount(0);
    BankWriter primaryWriter
      = new BankWriter("Kontoja primare", primaryAccount);
    BankAccount secondaryAccount = new BankAccount(0);
    BankWriter secondaryWriter
      = new BankWriter("Kontoja sekondare", secondaryAccount);
    AccountController2 controller
      = new AccountController2(reader, primaryWriter, primaryAccount,
            secondaryWriter, secondaryAccount);
    controller.processTransactions();
  }
}
```

Testimi i metodave dhe klasave

Secila metodë e secilës klasë testohet veç e veç.

Kur metoda përmban urdhëra `if` secili krah i secilit urdhër ekzekutohet së paku nga një test.

Në qoftë se klasa përmban fusha, atëherë testohet secila metodë e klasës në invariantat e paraqitjes së fushave.

Testimi i metodave dhe klasave – Vazhdim

```
public static void main(String[] args)
{ BankAccount a = new BankAccount(100);
  int i = new Integer(args[0]).intValue();
  boolean result = a.withdraw(i);
  System.out.println(result);
  System.out.println(a.getBalance());
}
```

Testimi i metodave dhe klasave – Vazhdim

```
public class BankAccount
{ private int balance;  // invariantë: balance >= 0

  public BankAccount(int amount) { balance = 0; } // dummy

  public boolean deposit(int amount)
  { boolean result = false;
    if ( amount < 0 )
    { JOptionPane.showMessageDialog(null, "Gabim"); }
    else { balance = balance + amount;
           result = true;
         }
    return result;
  }

  public boolean withdraw(int amount) { return true; } // dummy
  public int getBalance() { return 0; } // dummy
}
```