OOP dhe GUI

Programimi sipas interfejseve

Faton M. Berisha

fberisha@uni-pr.edu

Universiteti i Prishtinës

Fakulteti i Shkencave Matematike–natyrore

Referencat

Web sajte:

- `http://fberisha.netfirms.com`
- `www.cis.ksu.edu/santos/schmidt/ppj`
- `java.sun.com`

Literatura:

- D. Schmidt, *Programming principles in Java: Archtecture and Interfaces*.
- T. Jia, *Object oriented software development using Java: principles, patterns and frameworks*, Addison Wesley, 2000.
- I. Horton, *Beginning Java 2*, Wrox Press, 2000.

Programimi sipas interfejseve

Objektivat:

- Të mësohet Java konstruksioni `interface`, të cilin e shfrytëzojmë për të formuluar specifikacionet të cilat duhet t'i implementojnë klasat.
- Të mësohet Java konstruksioni `abstract class`, të cilin e shfrytëzojmë për të shkruar „klasë jokomplete“ të cilat janë pjesërisht specifikacione dhe pjesërisht implementime.
- Të mësohet Java konstruksioni `package`, të cilin e shfrytëzojmë për t'i grupuar klasat e një aplikacioni nën një emër.

Interfejset

Interfejs: specifikim i sjelljes (metodave) të pritura nga klasa. Java `interface` është një koleksion i emërtuar ballinash të metodave publike.

Sintaksa e Java interfejsit:

```
public interface EMRI
{ BALLINA_METODASH }
```

ku *BALLINA_METODASH* është varg ballinash të metodave të terminuara me pikëpresë.

Interfejset – Vazhdim

Procedura për konstruktimin e një interfejsi:

- Shkruarja e `interface` (si një fajl i veçantë) dhe kompilimi i tij.
- Shkruarja e klasës e cila implemenon (`implements`) interfejsin.
- Shkruarja e klasëve të cilat janë të parametrizuara (të lidhura) me interfejsin.

Interfejset – Vazhdim

Implementim interfejsi: shkruarja e klasës e cila përmban metodat ballinat e të cilave përputhen me ato të emërtuara në interfejs.

Në Java `interface` definon një emër tipi të dhënash, dhe kur një klasë implementon interfejsin emri i tipit të të dhënave të klasës është nëntip i emrit të interfejsit.

Interfejset – Vazhdim

```
/** Specifikon nje konto bankiere. */
public interface BankAccountSpecification
{ /** Deponon të holla në konto.
    *  @param amount - sasia për deponim
    *  @return true nëse deponimi i suksesshëm, false përndryshe */
  public boolean deposit(int amount);
  /** Tërheq të holla nga kontoja.
    *  @param amount - sasia për tërheqje
    *  @return true nëse tërheqja e suksesshme, false përndryshe */
  public boolean withdraw(int amount);
}
```

Interfejset – Vazhdim

```
/** Kryen pagesat e huas. */
public class MortgagePaymentCalculator
{ private BankAccountSpecification account;
  /** Konstruktori incializon kalkulatorin.
   *  @param a - kontoja bankiere nga e cila kryhen pagesat */
  public MortgagePaymentCalculator(BankAccountSpecification a)
  { account = a; }
  /** Kryen një pagesë huaje nga kontoja.
   *  @param amount - sasia e pagesës */
  public void makeMortgagePayment(int amount)
  { boolean ok = account.withdraw(amount);
    if (ok)
    { System.out.println("Pagesa është kryer:" + amount); }
    else { System.out.println("Pagesa nuk është kryer"); }
  }
}
```

Interfejset – Vazhdim

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

Interfejset – 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 )
  { System.out.println("Gabim: depositi " + amount);
  }
  else { balance += amount;
         result = true;
       }
  return result;
}
```

Interfejset – 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 )
  { System.out.println("Gabim: tërheqja " + amount); }
  else if ( amount > balance )
       { System.out.println("Gabim: tërheqja tejkalon balansin");
       }
  else { balance -= amount;
         result = true;
       }
  return result;
}
```

Interfejset – Vazhdim

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

Interfejset – Vazhdim

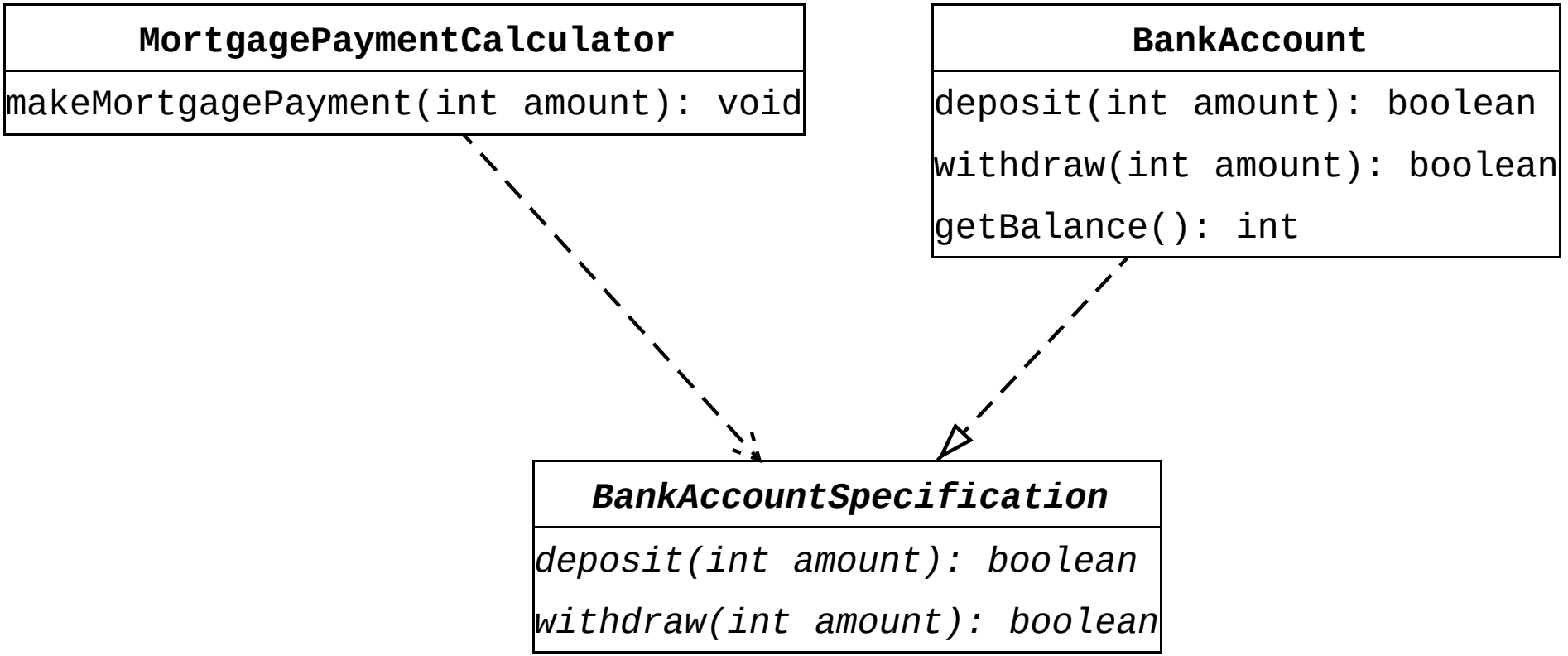


Figura 1. Diagrami i klasave të Java interfejsit

Interfejset – Vazhdim

```
public class MortgagePaymentTest
{ public static void main(String[] args)
  { BankAccount myAccount = new BankAccount(5000);
    MortgagePaymentCalculator calc =
      new MortgagePaymentCalculator(myAccount);
    calc.makeMortgagePayment(1000);
  }
}
```

Case study: bazat e të dhënave

```
/** Interfejs klase e cila mund të ruhet në një bazë të dhënash. */
public interface Record
{ /** Kthen çelësin i cili e identifikon
   *    në mënyrë unike regjistrimin.
   *  @return çelësi */
  public Key keyOf();
}
```

Case study: bazat e të dhënave – Vazhdim

```
/** Interfejs klase të vlerës çelës (identifikuese). */
public interface Key
{ /** Krahason në barazim veten me një çelës tjetër.
   *  @param m - çelësi tjetër
   *  @return true në qoftë se çelësi ka të njëjtën vlerë me m;
   *    false përndryshe */
  public boolean equals(Key m);
  /** Krahason në mosbarazim veten me një çelës tjetër.
   *  @param m - çelësi tjetër
   *  @return true në qoftë se çelësi është vlerë më e vogël se m;
   *    false përndryshe */
  public boolean lessthan(Key m);
}
```

Case study: bazat e të dhënave – Vazhdim

```
/** Implementon një bazë të dhënash. */
public class Database
{ private Record[] base;
  private int count;
  private int NOT_FOUND = -1;
  /** Inicializon bazën.
   *  @param initialSize - madhësia fillestare e bazës */
  public Database(int initialSize)
  { if ( initialSize > 0 )
    { base = new Record[initialSize]; }
    else { base = new Record[1]; }
    count = 0;
  }
```

Case study: bazat e të dhënave – Vazhdim

```
private int locationOf(Key k)
{ int result = NOT_FOUND;
  boolean found = false;
  int i = 0;
  while ( !found && i != base.length )
  { if ( base[i] != null && base[i].keyOf().equals(k) )
    { found = true;
      result = i;
    }
    else { i++; }
  }
  return result;
}
```

Case study: bazat e të dhënave – Vazhdim

```
/** Alokon një regjistrim mbështetur në çelësin.
 *  @param k - çelësi i regjistrimit të kërkuar
 *  @return rrokja e bazës (adresa e saj);
 *    null në qoftë se rrokja nuk është gjetur */
public Record find(Key k)
{ Record answer = null;
  int index = locationOf(k);
  if ( index != NOT_FOUND )
  { answer = base[index]; }
  return answer;
}
```

Case study: bazat e të dhënave – Vazhdim

```
/** Inserton një regjistrim në bazën e të dhënave.
 *  @param r - rrokja
 *  @return true në qoftë se rrokja është shtuar;
 *    false në qoftë se rrokja nuk është shtuar */
public boolean insert(Record r)
{ boolean success = false;
  if ( locationOf(r.keyOf()) == NOT_FOUND )
  { boolean foundSlot = false;
    int i = 0;
    while ( !foundSlot && i != base.length)
    { // deri më tani base[0]...base[i-1] janë të zëna
      if ( base[i] == null )
      { foundSlot = true; }
      else { i++; }
    }
```

Case study: bazat e të dhënave – Vazhdim

```
    if ( foundSlot )
    { base[i] = r; }
    else { Record[] temp = new Record[2 * base.length];
           for (int j = 0; j != count; j++)
           { // kopjohet përmbajtja e base në temp
             temp[j] = base[j];
           }
           base = temp;
           base[count] = r;
         }
    count++;
    success = true;
  }
  return success;
}
```

Case study: bazat e të dhënave – Vazhdim

```
  /** Fshin një regjistrim nga baza e të dhënave.
   *  @param k - çelësi i regjistrimit
   *  @return true në qoftë se rrokja është fshirë;
   *    false në qoftë se rrokja nuk është fshirë */
  public boolean delete(Key k)
  { boolean success = false;
    int index = locationOf(k);
    if ( index != NOT_FOUND )
    { base[index] = null;
      count--;
      success = true;
    }
    return success;
  }
}
```

Case study: bazat e të dhënave – Vazhdim

```
/** Modelon një person që di vetëm emrin dhe çelësin e vetë. */
public class BasicPerson implements Record
{ private Key k;
  private String who;
  /** Konstrukton regjistrimin e personit.
   *  @param name - emri i personit
   *  @param m - çelësi i personit */
  public BasicPerson(String name, Key m)
  { who = name;
    k = m;
  }
  /** Kthen emrin e personit. */
  public String nameOf()
  { return who; }

  public Key keyOf()
  { return k; }
}
```

Case study: bazat e të dhënave – Vazhdim

```
/** Modelon një çelës numër të plotë. */
public class IntegerKey implements Key
{ private int id;
  /** Konstrukton objektin çelës.
   *  @param n - numri i plotë i ruajtur si çelës */
  public IntegerKey(int n)
  { id = n; }
  /** Kthen vlerën numër të plotë të çelësit. */
  public int valOf()
  { return id; }
```

Case study: bazat e të dhënave – Vazhdim

```
  public boolean equals(Key anotherKey)
  { int m = ((IntegerKey)anotherKey).valOf();
    return id == m;
  }

  public boolean lessthan(Key anotherKey)
  { int m = ((IntegerKey)anotherKey).valOf();
    return id < m;
  }
}
```

Case study: bazat e të dhënave – Vazhdim

```
public static void main(String[] args)
{ Database db = new Database(4);
  BasicPerson filani =
    new BasicPerson("Filan Fisteku", new IntegerKey(1234));
  IntegerKey tringasKey = new IntegerKey(567);
  BasicPerson tringa =
    new BasicPerson("Tringa Kosova", tringasKey);
  boolean ok = db.insert(filani);
  ok = db.insert(tringa);
  Record p = db.find(tringasKey);
  System.out.println(((BasicPerson)p).nameOf());
}
```

Case study: bazat e të dhënave – Vazhdim

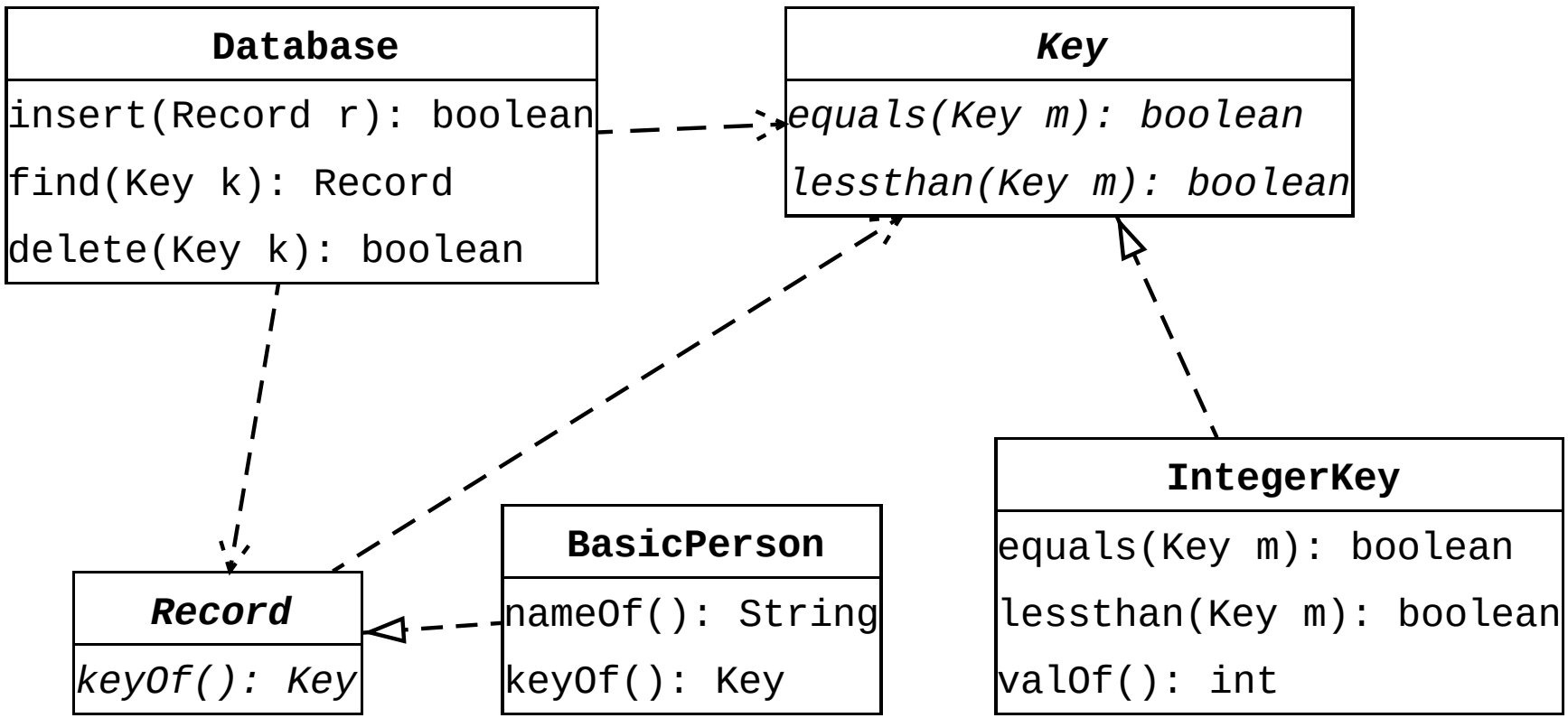


Figura 2. Diagrami i klasave me interfejse

Nëntipet dhe sjellja

Relacioni nëntip: relacion ndërmjet dy tipesh të të dhënave; shënojmë `C <= D` në kuptimin se `C` është nëntip i `D`, me domethënie se mund të shfrytëzohet vlerë e tipit `C` në çfarëdo konteksti ku pritet vlerë e tipit `D`.

P.sh., meqë `int <= double`, mund të vëhet

```
double d = 4.4 / 2;
```

ose për metodën

```
public double inverseOf(double d)
{ return 1.0 / d; }
```

është korrekt invokimi `inverseOf(2)`.

Nëntipet dhe sjellja – Vazhdim

Me zgjerimin ose implementimin e tipeve të referencës krijohet relacion i ri nëntip, p.sh.,

```
class MyPanel extends JPanel
```

dhe

```
class BasicPerson implements Record
```

krijojnë nëntipet `MyPanel <= JPanel` dhe `BasicPerson <= Record`.

Nëntipet dhe sjellja – Vazhdim

Vëjmë në aplikacionin e bazës së të dhënave

```
Database db = new Database(4);
IntegerKey filanisKey = new IntegerKey(1234);
BasicPerson filani = new BasicPerson("Filan Fisteku", filanisKey);
boolean ok = db.insert(filani);
```

Nëntipet dhe sjellja – Vazhdim

Database db == a1

IntegerKey filanisKey == a3

BasicPerson filani == a4

boolean success == true

a4 : BasicPerson

Key k == a3

String who = "Filan Fisteku"

...

a3 : IntegerKey

int id == 1234

...

a1 : Database

Record[] base == a2

int count == 1

public boolean insert(Record r) {...}

public Record find(Key k) {...}

public boolean delete(Key k) {...}

private int locationOf(Key k) {...}

a2 : Record[4]

0	1	2	3
a4	null	null	null

Nëntipet dhe sjellja – Vazhdim

Run-time tip i të dhënave: tipi i të dhënave i cili ruhet brenda një objekti kur ky të konstruktohet në memorien kompjuterike. Tipi i ruajtur i të dhënave është emri i klasës nga e cila është konstruktuar objekti.

Në vazhdim në aplikacionin e mësipërm mund të vëjmë

```
Record filaniAgain = db.find(filanisKey);
System.out.println(((BasicPerson)filaniAgain).nameOf());
```

Invokimi `filaniAgain.nameOf()` është jokorrekt.

Nëntipet dhe sjellja – Vazhdim

`instanceof`*:* operacion në Java i cili ekzaminon run-time tipin e objektit.

Sintaksa:
SHPREHJA `instanceof` *TIPI*

Semantika: kthen `true` atëherë dhe vetëm atëherë kur run-time tipi i të dhënave i *SHPREHJA* është nëntip i *TIPI*.

```
Record aRecord = db.find(aKey);
if ( aRecord instanceof BasicPerson )
{ System.out.println(((BasicPerson)aRecord).nameOf()); }
else { System.out.println("Tip i panjohur regjistrimesh"); }
```

Nëntipet dhe sjellja – Vazhdim

```
/** Modelon një çelës string. */
public class StringKey implements Key
{ private String id;

  public StringKey(String s)
  { id = s; }

  public String valOf()
  { return id; }

  public boolean equals(Key anotherKey)
  { ... }

  public boolean lessthan(Key anotherKey)
  { ... }
}
```

Nëntipet dhe sjellja – Vazhdim

```
public boolean equals(Key anotherKey)
{ boolean answer;
  if ( anotherKey instanceof IntegerKey )
  { int m = ((IntegerKey)anotherKey).valOf();
    answer = (id == m);
  }
  else { answer = false; }
  return answer;
}
```

Implementimi dhe trashigimia: një lojë aventure

```
/** Definon sjelljen e një dhome. */
public interface RoomBehavior
{ /** Lejon lojtarin të hyjë në dhomë.
   *  @param p - lojtari që dëshiron të hyjë
   *  @return se a ka pasur sukses hyrja */
  public boolean enter(PlayerBehavior p);
  /** Nxjerr lojtarin nga dhoma.
   *  @param p - lojtari që dëshiron të dalë */
  public void exit(PlayerBehavior p);
  /** Kthen idenitetin e lojtarit i cili ka zënë dhomën.
   *  @return adresën e lojtarit që ndodhet në dhomë */
  public PlayerBehavior occupantOf();
}
```

Implementimi dhe trashigimia: një lojë aventure – Vazhdim

```
/** Definon sjelljen e një lojtari. */
public interface PlayerBehavior
{ /** Lejon lojtarin të thotë një fjalë.
   *  @return fjala */
  public String speak();
  /** Përpiqet të hyjë në dhomë dhe ta esplorojë atë.
   *  @param r - dhoma që do të eksplorohet
   *  @return se a ka pasur sukses hyrja në dhomën */
  public boolean explore(RoomBehavior r);
}
```

Implementimi dhe trashigimia: një lojë aventure – Vazhdim

```
/** Modelon dhomë që mund të ketë një banor. */
public class BasicRoom implements RoomBehavior
{ private PlayerBehavior occupant;
  private String roomName;
  private String secretWord;
  /** Ndërton dhomën.
   *  @param name - emri i dhomës
   *  @param password - fjala sekrete për të hyrë në dhomën */
  public BasicRoom(String name, String password)
  { occupant = null;
    roomName = name;
    secretWord = password;
  }
```

Implementimi dhe trashigimia: një lojë aventure – Vazhdim

```
  public boolean enter(PlayerBehavior p)
  { boolean answer = false;
    if ( occupant == null && secretWord.equals(p.speak()) )
    { occupant = p;
      answer = true;
    }
    return answer;
  }

  public void exit(PlayerBehavior p)
  { if ( occupant == p)
    { occupant = null; }
  }

  public PlayerBehavior occupantOf()
  { return occupant; }
}
```

Implementimi dhe trashigimia: një lojë aventure – Vazhdim

```
/** Modelon një lojtar që eksploron dhoma. */
public class Explorer implements PlayerBehavior
{ private String playerName;
  private String password;
  private RoomBehavior where;
  /** Ndërton lojtarin.
   *  @param name - emri i lojtarit
   *  @param word - pasvordi të cilin mund ta thotë lojtari */
  public Explorer(String name, String word)
  { playerName = name;
    password = word;
    where = null;
  }

  public String speak()
  { return password; }
```

Implementimi dhe trashigimia: një lojë aventure – Vazhdim

```
/** Bën që lojtari ta lëjë dhomën në të cilën ndodhet. */
public void exitRoom()
{ if ( where != null)
  { where.exit(this);
    where = null;
  }
}

public boolean explore(RoomBehavior r)
{ if ( where != null )
  { exitRoom(); }
  boolean wentInside = r.enter(this);
  if ( wentInside )
  { where = r; }
  return wentInside;
}
```

Implementimi dhe trashigimia: një lojë aventure – Vazhdim

```
  /** Kthen dhomën që është e zënë nga lojtari. */
  public RoomBehavior locationOf()
  { return where; }
}
```

Implementimi dhe trashigimia: një lojë aventure – Vazhdim

```
RoomBehavior[] groundFloor = new RoomBehavior[4];
groundFloor[0] = new BasicRoom("kuzhina", "brumi");
groundFloor[3] = new BasicRoom("dhoma e ditës", "çaji");
Explorer indi = new Explorer("Indana Jones", "çaji");
Explorer xhyma = new Explorer("Xhymshit Fisteku", "domatja");
boolean success = indi.explore(groundFloor[3]);
```

Implementimi dhe trashigimia: një lojë aventure – Vazhdim

RoomBehavior[] groundFloor == a1

Explorer indi == a4

Explorer xhyma == a5

a1 : RoomBehavior[4]

0	1	2	3
a2	null	null	a3

a2 : BasicRoom

...

a5 : Explorer

...

a3 : BasicRoom

PlayerBehavior occupant == null

String roomName == "dhoma e ditës"

String secretWord == "çaji"

boolean enter(PlayerBehavior p)

void exit(PlayerBehavior p)

PlayerBehavior occupantOf()

a4 : Explorer

String playerName == "Indana Jones"

String password == "çaji"

RoomBehavior where == null

String speak()

void exitRoom()

boolean explore(RoomBehavior r)

RoomBehavior locationOf()

Implementimi dhe trashigimia: një lojë aventure – Vazhdim

RoomBehavior[] groundFloor == a1

Explorer indi == a4

Explorer xhyma == a5

boolean success == true

a1 : RoomBehavior[4]

0	1	2	3
a2	null	null	a3

a2 : BasicRoom

...

a5 : Explorer

...

a3 : BasicRoom

PlayerBehavior occupant == a4

String roomName == "dhoma e ditës"

String secretWord == "çaji"

boolean enter(PlayerBehavior p)

void exit(PlayerBehavior p)

PlayerBehavior occupantOf()

a4 : Explorer

String playerName == "Indana Jones"

String password == "çaji"

RoomBehavior where == a3

String speak()

void exitRoom()

boolean explore(RoomBehavior r)

RoomBehavior locationOf()

Implementimi i interfejseve të shumfishta

```
/** Specifikon një objekt thesar. */
public interface TreasureProperty
{ /** Shpjegon çfarë është thesari.
   *  @return shpjegimi */
  public String contentsOf();
}

/** Specifikon mënyrën e një objekti për të dorëzuar thesarin. */
public interface Treasury
{ /** Dorëzon thesarin.
   *  @param p - lojtari i cili kërkon thesarin
   *  @return thesari (ose null në qoftë se thesari mungon) */
  public TreasureProperty yieldTreasure(PlayerBehavior p);
}
```

Implementimi i interfejseve të shumfishta – Vazhdim

```
public class Vault implements RoomBehavior, Treasury
{ private PlayerBehavior occupant;
  private String roomName;
  private String secretWord;
  private TreasureProperty valuable;
  /** Ndërton dhomën.
   *  @param name - emri i dhomës
   *  @param password - fjala sekrete për të hyrë në dhomën
   *  @param item - thesari i cili ruhet në dhomën */
  public Vault(String name, String password, TreasureProperty item)
  { occupant = null;
    roomName = name;
    secretWord = password;
    valuable = item;
  }
```

Implementimi i interfejseve të shumfishta – Vazhdim

```
  public boolean enter(PlayerBehavior p)
  { // ... sikur më parë
  }
  public void exit(PlayerBehavior p)
  { // ... sikur më parë
  }
  public PlayerBehavior occupantOf()
  { // ... sikur më parë
  }
  public TreasureProperty yieldTreasure(PlayerBehavior p)
  { TreasureProperty answer = null;
    if ( p == occupant )
    { answer = valuable;
      valuable = null;
    }
    return answer;
  }
}
```

Nënklasat dhe trashigimia

Sintaksa:

```
class C2 extends C1
{ TRUPI }
```

Semantika: klasa `class C2` merr (trashigon) tërë strukturën brenda `class C1` si dhe strukturën e definuar në *`TRUPI`*. Vëhet relacioni `C2 <= C1`.

Klasa `C2` quhet *nënklasë (subclass)* e *mbiklasës (superclass)* `C1`.

Konstruktori `super`*:* urdhëri i cili invokon metodën konstruktor të superklasës përbrenda metodës konstruktor të nënklasës së saj.

Nënklasat dhe trashigimia – Vazhdim

```
/** Modelon dhomë ku ruhet thesar. */
public class VaultRoom extends BasicRoom implements Treasury
{ private TreasureProperty valuable;
  /** Ndërton dhomën.
   *  @param name - emri i dhomës
   *  @param password - fjala sekrete për të hyrë në dhomën
   *  @param item - thesari i cili ruhet në dhomën */
  public VaultRoom(String name, String password,
      TreasureProperty item)
  { super(name, password);
    valuable = item;
  }
```

Nënklasat dhe trashigimia – Vazhdim

```
  public TreasureProperty yieldTreasure(PlayerBehavior p)
  { TreasureProperty answer = null;
    if ( p == occupantOf() )
    { answer = valuable;
      valuable = null;
    }
    return answer;
  }
}
```

Nënklasat dhe trashigimia – Vazhdim

```
BasicRoom a = new BasicRoom("e ditës", "Tung!");
TreasureProperty diamond = new Jewel("diamant");
VaultRoom b = new VaultRoom("podrumi", "Hapu Sezam!", diamond);

/** Modelon një gur të çmueshëm. */
public class Jewel implements TreasureProperty
{ private String jewelName;
  /** Konstrukton gurin e çmueshëm.
   *  @param name - emri i gurit */
  public Jewel(String name)
  { jewelName = name; }

  public String contentsOf()
  { return jewelName; }
}
```

Nënklasat dhe trashigimia – Vazhdim

BasicRoom a == a1

TreasureProperty diamond == a2

VaultRoom b == a3

a1 : BasicRoom

PlayerBehavior occupant == null

String roomName == "e ditës"

String secretWord == "Tung!"

boolean enter(PlayerBehavior p)

void exit(PlayerBehavior p)

PlayerBehavior occupantOf()

a2 : Jewel

String jewelName == "diamant"

String contentsOf()

a3 : VaultRoom

PlayerBehavior occupant == null

String roomName == "podrumi"

String secretWord == "Hapu Sezam!"

boolean enter(PlayerBehavior p)

void exit(PlayerBehavior p)

PlayerBehavior occupantOf()

TreasureProperty valuable == a2

TreasureProperty yieldTreasure(PlayerBehavior p)

Nënklasat dhe trashigimia – Vazhdim

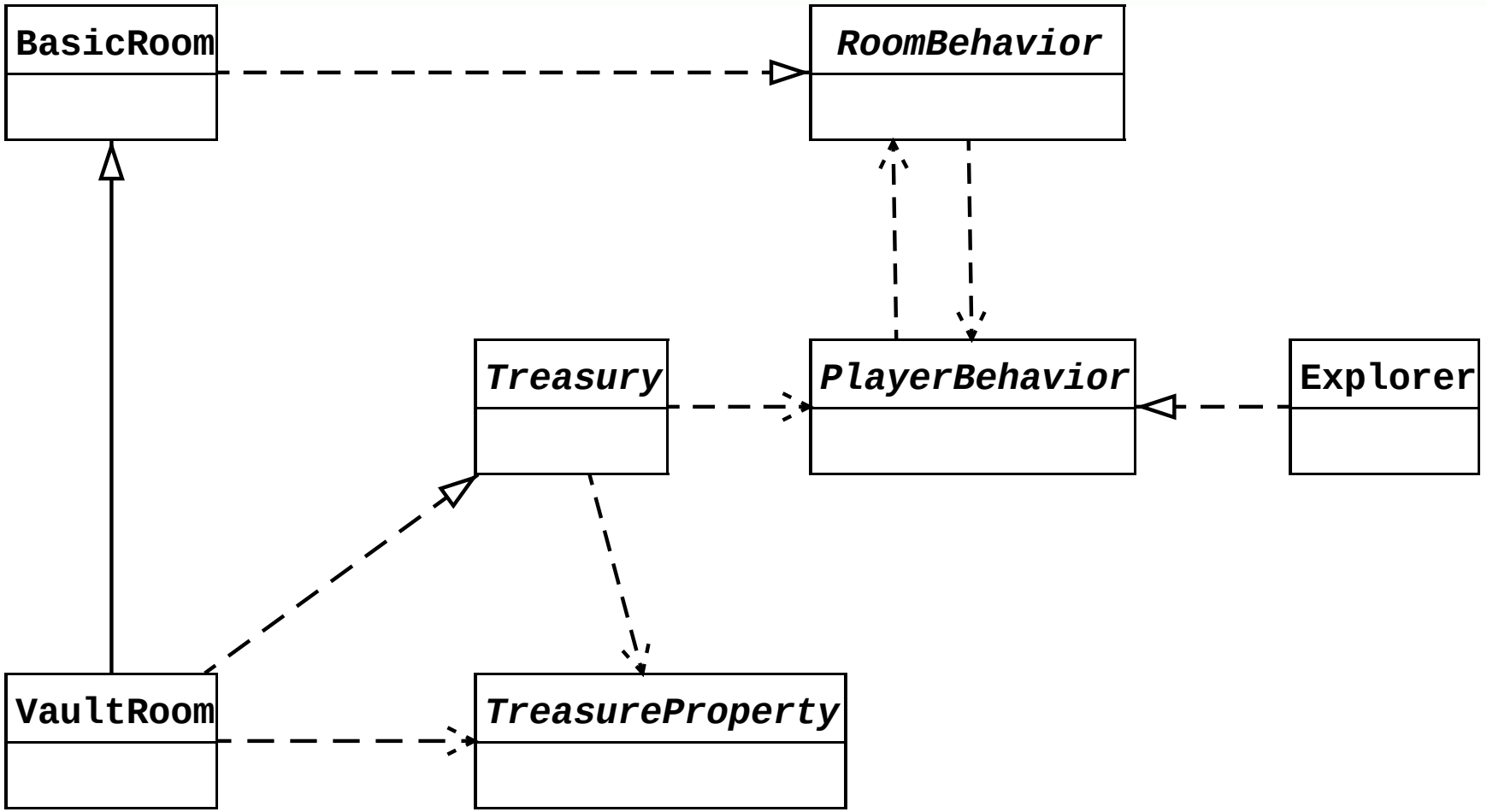


Figura 3. Diagrami i modelit të lojës së aventurës

Interfejsat dhe trashigimia

```
/** Definon sjelljen e një dhome që mban thesar. */
public interface TreasuryBehavior extends RoomBehavior
{ /** Dorëzon thesarin.
   *  @param p - lojtari i cili kërkon thesarin
   *  @return thesari (ose null në qoftë se thesari mungon) */
  public TreasureProperty yieldTreasure(PlayerBehavior p);
}

public class VaultRoom2 extends BasicRoom
    implements Treasury, TreasuryBehavior
{ ... }

public class VaultRoom2 extends BasicRoom
    implements TreasuryBehavior
{ ... }
```

Klasat abstrakte

Klasë abstrakte: klasë së cilës i mungojnë disa nga metodat e veta; metodat që mungojnë vëhen në pah me ballinë e cila përmban fjalën kyçe `abstract`.

Metodë abstrakte: metodë pa trup e cila ndodhet në një klasë abstrakte. Trupi i metodës shkruhet në një nënklasë të klasës abstrakte.

Klasë konkrete: klasë „normale“, të gjitha metodat e së cilës kanë trupa.

Klasat abstrakte – Vazhdim

```
/** Definon sjelljen e pritur nga lojtari me karta. */
public interface CardPlayerBehavior
{ /** Kthen se a dëshiron lojtari edhe një kartë të re. */
  public boolean wantsCard();
  /** Pranon një kartë.
   *  @param c - karta që pranohet  */
  public void receiveCard(Card c);
  /** Afishon dorën e lojtarit.
   *  @return një varg i cili përmban kartat e dorës */
  public Card[] showCards();
}
```

Klasat abstrakte – Vazhdim

```
/** Modelon një formë abstrakte të lojtarit me karta. */
public abstract class CardPlayer implements CardPlayerBehavior
{ private Card[] hand;
  private int count;
  /** Ndërton lojtarin.
   *  @param maxCards - numri maksimal i kartave */
  public CardPlayer(int maxCards)
  { hand = new Card[maxCards];
    count = 0;
  }

  public abstract boolean wantsCard();
```

Klasat abstrakte – Vazhdim

```
  public void receiveCard(Card c)
  { hand[count] = c;
    count++;
  }

  public Card[] showCards()
  { Card[] answer = new Card[count];
    for ( int i = 0; i != count; i++ )
    { answer[i] = hand[i]; }
    return answer;
  }
}
```

Klasat abstrakte – Vazhdim

```
import javax.swing.*;
/** Modelon një lojtar kartash njeri. */
public class HumanPlayer extends CardPlayer
{ /** Ndërton lojtarin.
   *  @param maxCards - numri maksimal i kartave */
  public HumanPlayer(int maxCards)
  { super(maxCards); }

  public boolean wantsCard()
  { String response = JOptionPane.showInputDialog
      ("Dëshironi edhe një kartë (P/J)?");
    return response.equals("P") || response.equals("p");
  }
}
```

Klasat abstrakte – Vazhdim

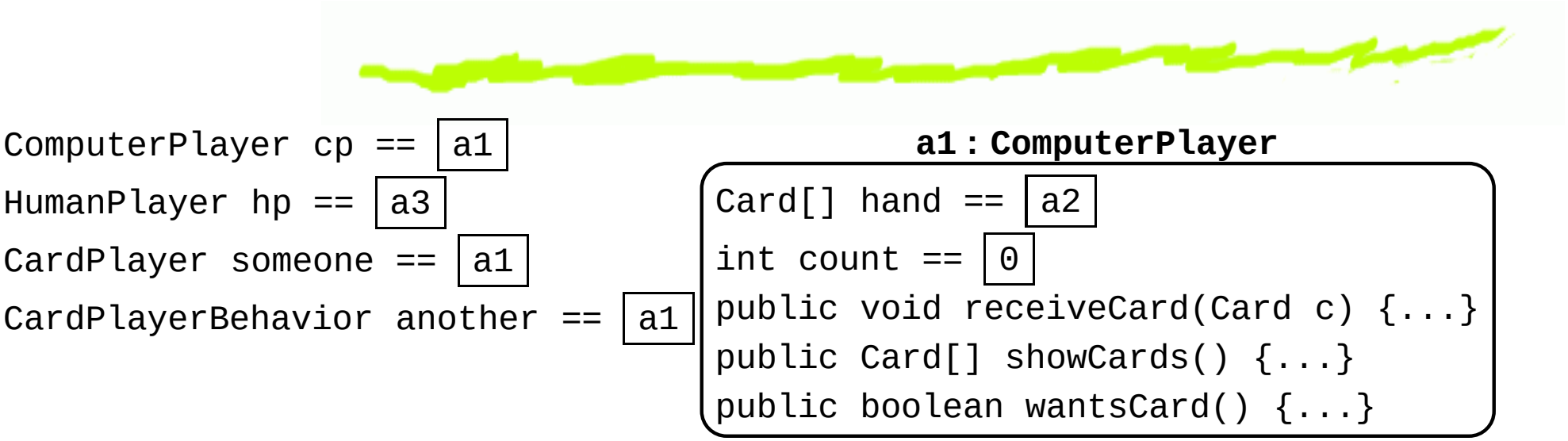
```
/** Modelon një lojtar kartash të kompjuterizuar. */
public class ComputerPlayer extends CardPlayer
{ /** Ndërton lojtarin.
    *  @param maxCards - numri maksimal i kartave */
  public ComputerPlayer(int maxCards)
  { super(maxCards); }

  public boolean wantsCard()
  { boolean decision = false;
    Card[] myHand = showCards();
    // ... Urdhërat të cilat ekzmainojnë myHand
    //       dhe llogarisin vendimin decision
    return decision;
  }
}
```

Klasat abstrakte – Vazhdim

```
ComputerPlayer cp = new ComputerPlayer(3);
HumanPlayer hp = new HumanPlayer(3);
CardPlayer someone = cp;
CardPlayerBehavior another = someone;
```

Klasat abstrakte – Vazhdim



Klasat abstrakte – Vazhdim

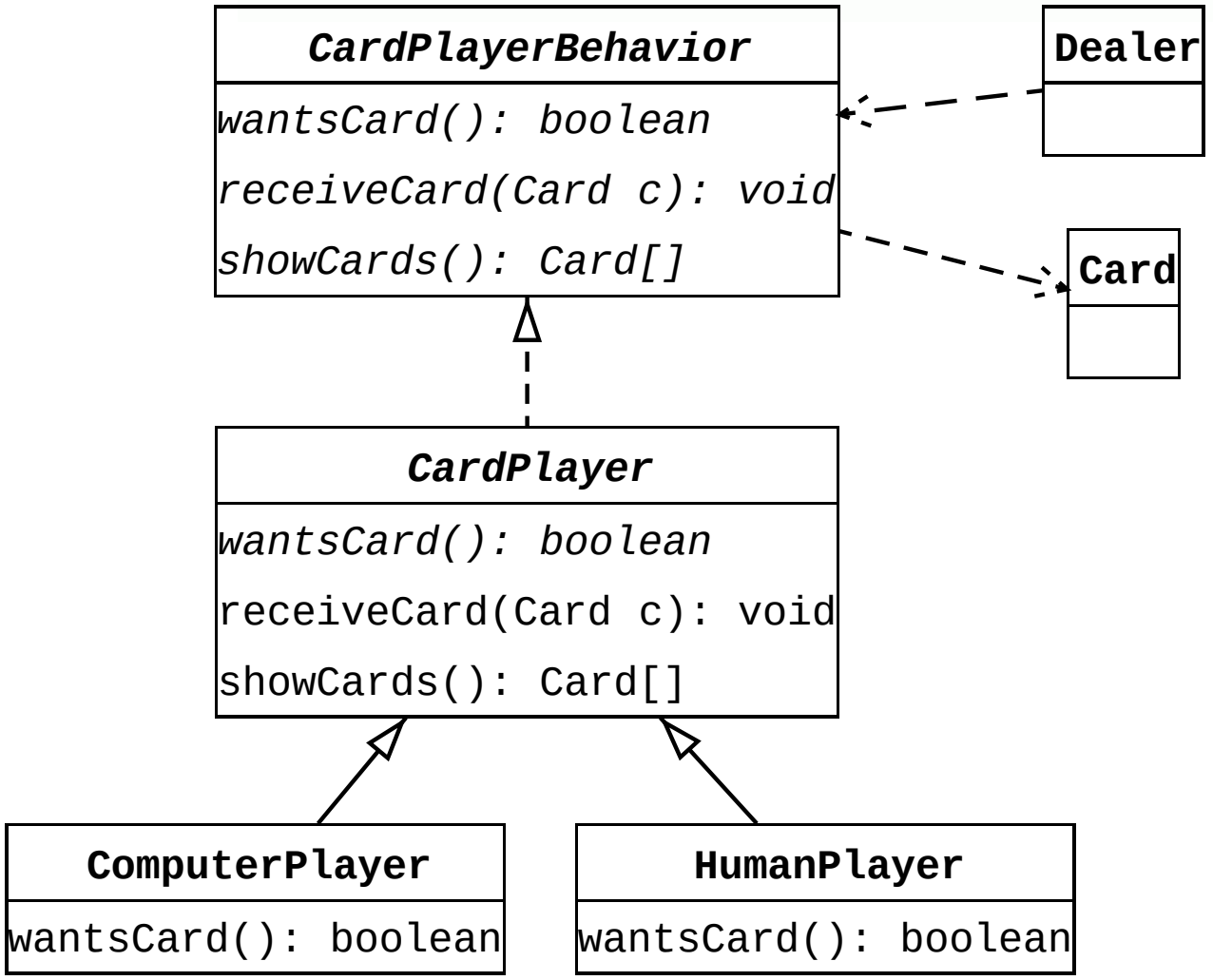


Figura 4. Arkitektura e lojës së kartave

Klasat abstrakte dhe kierarkitë

Kierarki klase: koleksion klasash abstrakte dhe konkrete, të renditura sipas relacioneve të tyre nën/mbiklasë, zakonisht i paraqitur si strukturë druri.

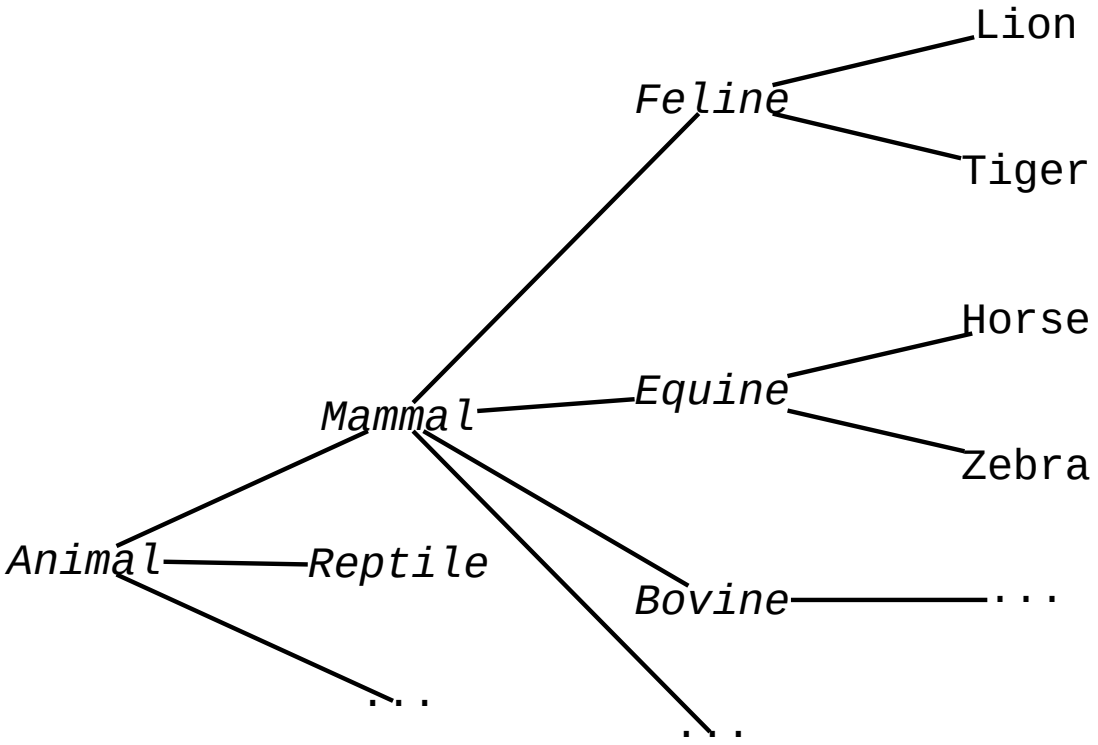


Figura 5. Kierarkia e shtazëve

Klasat abstrakte dhe kierarkitë – Vazhdim

```
public abstract class Animal
{ //... fushat dhe metodat të cilat e përshkruajnë
  //      një shtazë
}

public abstract class Mammal extends Animal
{ //... fushat dhe metodat specifike për gjitarët
}

public abstract class Equine extends Mammal
{ //... fushat dhe metodat specifike për thundrorët
}

public class Horse extends Equine
{ //... fushat dhe metodat specifike për kuajt
}
```

Klasat abstrakte dhe kierarkitë – Vazhdim

```
Horse ati = new Horse(...);
Equine aSpecimen = ati;
```

Kornizat dhe klasat abstrakte

Kornizë (framework): koleksion klasash të disenjuara për t'u zgjeruar me vetëm disa klasa shtesë për të konstruktuar një aplikacion të plotë në një domen problemi.

P.sh., pakot `java.awt` dhe `javax.swing` formojnë një kornizë për aplikacione grafike; duke zgjeruar `JPanel` shfrytëzuesi „plotëson" kornizën dhe krijon dritare grafike.

Nëntipet dhe nënklasat

Dallimet ndërmjet interfesjseve dhe klasave abstrakte:

- Një interfejs definon një specifikim sjelljeje ose një „pikë lidhjeje“ të cilën klasët e shfrytëzojnë për t’u lidhur ndërmjet veti. Kur një klasë e implementon një interfejs klasa ofron sjelljen e premtuar nga interfejsi.
- Një klasë abstrakte ofron kodet – implementimet – për disa metoda dhe lëshon kodimin e tjerave. Kur një klasë zgjeron një klasë abstrakte ajo trashëgon kodimet ekzistuese dhe ofron kodimet që mungojnë.

Klasa `Object` dhe mbështjellësit

`class Object`*:* klasë e paradefinuar në Java e cila automatikisht është mbiklasë e të gjitha klasave tjera.

```
public class Pair
{ Object[] r = new Object[2];

  public Pair(Object o1, Object o2)
  { r[0] = o1;
    r[1] = o2;
  }

  public Object getFirst()
  { return r[0]; }

  public Object getSecond()
  { return r[1]; }
}
```

Klasa `Object` dhe mbështjellësit – Vazhdim

```
Pair p = new Pair(new JFrame(), new int[3]);
Object item = p.getFirst();
if ( item instanceof JFrame )
{ ((JFrame)item).setVisible(true); }
```

Klasa `Object` dhe mbështjellësit – Vazhdim

Klasë mbështjellëse (wrapper): klasë qëllimi i së cilës është të përmbajë një vlerë të vetme primitive, duke lejuar kështu vlerën primitive të përdoret sikur të ishte objekt.

```
Integer wrappedInt = new Integer(1);
Pair p = new Pair(wrappedInt, new int[3]);
Object item = p.getFirst();
if ( item instanceof Integer )
{ System.out.println( ((Integer)item).intValue() ); }
```

Klasa `Object` dhe mbështjellësit – Vazhdim

Meqë çdo objekt ndërtohet nga një klasë e cila zgjëron klasën `Object`, çdo objekt trashëgon nga `class Object` metodën `toString`, e cila kthen paraqitjen si string të „identitetit“ të objektit.

```
Integer wrappedInt = new Integer(1);
String s = "abc";
Pair p = new Pair(wrappedInt, s);
System.out.println(p.getFirst().toString());
System.out.println(p.getSecond().toString());
System.out.println(p.toString());
```

Pakot

Pako (package): koleksion klasash të cilat janë grupuar së bashku në një folder dhe janë etiketuar me një emër të përbashkët.

```
package bank;
/** Specifikon nje konto bankiere. */
public interface BankAccountSpecification
{ / ... njësoj sikurse më parë
}
```

```
package bank;
import javax.swing.*;
/** Menagjon një konto. */
public class BankAccount
{ / ... njësoj sikurse më parë
}
```

Pakot – Vazhdim

Procedura e krijimit të një pakoje:

- Krijohet një folder, p.sh., `bank`.
- Vëhet, si rresht i parë në secilën klasë, urdhëri `package`, p.sh., `package bank`.
- Kompilohet secila klasë, p.sh., `javac bank\BankAccount.java`.
- Ekzekutohet klasa startuese, p.sh., `java bank.MortgagePaymentApplication`.

Pakot – Vazhdim

Në qoftë se aplikacione tjera e importojnë një pako, p.sh., `bank`, atëherë lokacioni i pakos i bëhet me dije Java kompilatorit me anë të *classpath*.

```
import bank.*;
/** Menagjon një konto çekësh. */
public class CheckingAccount
{ private BankAccount myAccount = new BankAccount();
  private MortgagePaymentCalculator calculator
    = new MortgagePaymentCalculator(myAccount);
  / ...
}
```