OOP & GUI

Provimi periodik 1, Forma: A

Emri: _____

ID (Nr. dosjes): _____

Drejtimi: _____

Data: _____

1. Vëni konvertimet eksplicite (casting) aty ku ka nevojë ashtu që vargu vijues i instruksioneve të kompilohet në mënyrë korrekte.

```
I x = new E();
x.f(6);
if ( x instanceof E )
{ x.g(); }
E y = x;
y.g();
```

2. Është dhënë interfejsi

```
public interface Convertable
{ public int double2int(double d); }
```

A e implementon në mënyrë korrekte këtë interfejs klasa vijuese? Arsyetoni përgjegjjen.

```
public class C1 implements Convertable
{ private double x;
  public C1(double a)
  { x = a; }
  public int double2int(double y)
  { return (int)(x + y); }
}
```

Pyetjet 3–6 përbëjnë një tërësi. Për t'u përgjegjur në to, shqyrtoni klasat vijuese.

```
/** Modelon një kartë loje. */
public class Card
{ public static final String SPADES = "spades";
  public static final String HEARTS = "hearts";
  public static final String DIAMONDS = "diamonds";
  public static final String CLUBS = "clubs";
  public static final int SUIT_SIZE = 13;
  public static final int ACE = 1;
  public static final int JACK = 11;
  public static final int QUEEN = 12;
  public static final int KING = 13;

  private String suit;
  private int count;

  /** Vë vlerën dhe ngjyrën e kartës.
   *  @param s - ngjyra
   *  @param c - vlera  */
  public Card(String s, int c)
  { suit = s;
    count = c;
```

```
  }

  /** Kthen ngjyrën e kartës. */
  public String suitOf()
  { return suit; }

  /** Kthen vlerën e kartës. */
  public int countOf()
  { return count; }
}

/** Modelon një shpil kartash. */
public class CardDeck
{ private Card[] deck = new Card[4 * Card.SUIT_SIZE];
  private int count;
  /** Krijon shpil të ri me të gjitha kartat. */
  public CardDeck()
  { createSuit(Card.SPADES);
    createSuit(Card.HEARTS);
    createSuit(Card.DIAMONDS);
    createSuit(Card.CLUBS);
  }
  /** Merr një kartë të re nga shpili.
   * @return  një kartë të re (ose null nëse nuk ka karta) */
  public Card newCard()
  { Card nextCard = null;
    if ( count == 0 )
    { System.out.println("Gabim: nuk kanë mbetur karta"); }
    else
    { int index = (int)(Math.random() * count);
      nextCard = deck[index];
      for (int i = index + 1; i != count; i = i + 1)
      // kartat nga index+1...i-1 janë lëvizur në të majtë
      { deck[i-1] = deck[i]; }
      count = count - 1;
    }
    return nextCard;
  }
  /** Tregon se a ka akoma karta në shpil.
   * @return  se a është shpili jo i zbrazët */
  public boolean moreCards()
  { return count > 0; }

  private void createSuit(String s)
  { for (int i = 1; i <= Card.SUIT_SIZE; i = i + 1)
    { deck[count] = new Card(s, i);
      count = count + 1;
    }
  }
}

/** 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();
}

/** 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 që mund t'i mbajë */
  public CardPlayer(int maxCards)
  { hand = new Card[maxCards];
    count = 0;
  }

  public abstract boolean wantsCard();

  public void receiveCard(Card c)
  { hand[count] = c;
    count = count + 1;
  }

  public Card[] showCards()
  { Card[] answer = new Card[count];
    for (int i = 0; i != count; i = i + 1)
    { answer[i] = hand[i]; }
    return answer;
  }
}

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");
  }
}
```

3. Shkruani `class Dealer` sipas specifikimit të dhënë në tabelën:

`class Dealer`	modelon një shpërndarës kartash
Metodat:	
`dealTo(CardPlayerBehavior p, int n)`	lojtarit `p` i jep `numCards` karta, një nga një
Kolaboratorët:	`CardPlayerBehaviour`, `CardDeck`

Tab. 1: Specifikimi i interfejsit për shpërndarësin e kartave

4. Shkruani një aplikacion, `TestDealer`, i cili krijon 4 lojtarë dhe një objekt shpërndarës kartash, i cili pastaj secilit nga lojtarët i ndan nga 5 karta.

5. Modifikoni aplikacionin nga detyra paraprake ashtu që në dritaren komanduese të afishohet dora (kartat të cilat mban) e secilit nga lojtarët.

6. Vizatoni diagramin e klasave për aplikacionin nga detyra paraprake.

Çelësi i provimit A

1. Vëni konvertimet eksplicite (casting) aty ku ka nevojë ashtu që vargu vijues i instruksioneve të kompilohet në mënyrë korrekte.

   ```
   I x = new E();
   x.f(6);
   if ( x instanceof E )
   { x.g(); }
   E y = x;
   y.g();
   ```

   Përgjegjja:

   ```
   I x = new E();
   x.f(6);
   if ( x instanceof E )
   { ((E)x).g(); }
   E y = (E)x;
   y.g();
   ```

2. Është dhënë interfejsi

   ```
   public interface Convertable
   { public int double2int(double d); }
   ```

   A e implementon në mënyrë korrekte këtë interfejs klasa vijuese? Arsyetoni përgjegjjen.

   ```
   public class C1 implements Convertable
   { private double x;
     public C1(double a)
     { x = a; }
     public int double2int(double y)
     { return (int)(x + y); }
   }
   ```

   Përgjegjja: Po. Është implementuar metoda e listuar në interfejsin.

Pyetjet 3–6 përbëjnë një tërësi. Për t'u përgjegjur në to, shqyrtoni klasat vijuese.

```
/** Modelon një kartë loje. */
public class Card
{ public static final String SPADES = "spades";
  public static final String HEARTS = "hearts";
  public static final String DIAMONDS = "diamonds";
  public static final String CLUBS = "clubs";
  public static final int SUIT_SIZE = 13;
  public static final int ACE = 1;
  public static final int JACK = 11;
  public static final int QUEEN = 12;
  public static final int KING = 13;

  private String suit;
```

```
  private int count;

  /** Vë vlerën dhe ngjyrën e kartës.
   *  @param s - ngjyra
   *  @param c - vlera  */
  public Card(String s, int c)
  { suit = s;
    count = c;
  }

  /** Kthen ngjyrën e kartës. */
  public String suitOf()
  { return suit; }

  /** Kthen vlerën e kartës. */
  public int countOf()
  { return count; }
}

/** Modelon një shpil kartash. */
public class CardDeck
{ private Card[] deck = new Card[4 * Card.SUIT_SIZE];
  private int count;
  /** Krijon shpil të ri me të gjitha kartat. */
  public CardDeck()
  { createSuit(Card.SPADES);
    createSuit(Card.HEARTS);
    createSuit(Card.DIAMONDS);
    createSuit(Card.CLUBS);
  }
  /** Merr një kartë të re nga shpili.
   * @return  një kartë të re (ose null nëse nuk ka karta) */
  public Card newCard()
  { Card nextCard = null;
    if ( count == 0 )
    { System.out.println("Gabim: nuk kanë mbetur karta"); }
    else
    { int index = (int)(Math.random() * count);
      nextCard = deck[index];
      for (int i = index + 1; i != count; i = i + 1)
      // kartat nga index+1...i-1 janë lëvizur në të majtë
      { deck[i-1] = deck[i]; }
      count = count - 1;
    }
    return nextCard;
  }
  /** Tregon se a ka akoma karta në shpil.
   * @return  se a është shpili jo i zbrazët */
  public boolean moreCards()
  { return count > 0; }

  private void createSuit(String s)
  { for (int i = 1; i <= Card.SUIT_SIZE; i = i + 1)
    { deck[count] = new Card(s, i);
```

```
      count = count + 1;
    }
  }
}

/** 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();
}

/** 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 që mund t'i mbajë */
  public CardPlayer(int maxCards)
  { hand = new Card[maxCards];
    count = 0;
  }

  public abstract boolean wantsCard();

  public void receiveCard(Card c)
  { hand[count] = c;
    count = count + 1;
  }

  public Card[] showCards()
  { Card[] answer = new Card[count];
    for (int i = 0; i != count; i = i + 1)
    { answer[i] = hand[i]; }
    return answer;
  }
}

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");
  }
}
```

3. Shkruani `class Dealer` sipas specifikimit të dhënë në tabelën:

`class Dealer`	modelon një shpërndarës kartash
Metodat:	
`dealTo(CardPlayerBehavior p, int n)`	lojtarit `p` i jep `numCards` karta, një nga një
Kolaboratorët:	`CardPlayerBehaviour`, `CardDeck`

Tab. 2: Specifikimi i interfejsit për shpërndarësin e kartave

Përgjegjja:

```
/** Modelon një shpërndarës kartash. */
public class Dealer
{ private CardDeck deck;

  /** Konstrukton shpërndarësin. */
  public Dealer()
  { deck = new CardDeck(); }

  /** I jep lojtarit të dhënë një numër të dhënë kartash,
   *  një nga një.
   *  @param p - lojtari që pranon kartat
   *  @param numCards - numri i kartave të ndara
   */
  public void dealTo(CardPlayerBehavior p, int numCards)
  { boolean processing = true;
    int i = 0;
    while ( i < numCards  &&  deck.moreCards() )
    { p.receiveCard(deck.newCard());
      i++;
    }
  }
}
```

4. Shkruani një aplikacion, `TestDealer`, i cili krijon 4 lojtarë dhe një objekt shpërndarës kartash, i cili pastaj secilit nga lojtarët i ndan nga 5 karta.

Përgjegjja:

```
public class TestDealer {
  public static void main(String[] args) {
    Dealer dealer = new Dealer();
    CardPlayerBehavior[] cp = new CardPlayerBehavior[4];
    for ( int i = 0; i < cp.length; i++ )
    { cp[i] = new ComputerPlayer(5);
      dealer.dealTo(cp[i], 5);
    }
  }
}
```

5. Modifikoni aplikacionin nga detyra paraprake ashtu që në dritaren komanduese të afishohet dora (kartat të cilat mban) e secilit nga lojtarët.

Përgjegjja:

```
public class TestDealer {
  public static void main(String[] args) {
    // ... njësoj sikur më parë

    for ( int i = 0; i < cp.length; i++ )
    { Card[] hand = cp[i].showCards();
      System.out.println("Dora e lojtarit " + (i + 1));
      for ( int j = 0; j < hand.length; j++ )
      {
        System.out.println(hand[j].suitOf() + " "
          + hand[j].countOf());
      }
    }
  }
}
```

6. Vizatoni diagramin e klasave për aplikacionin nga detyra paraprake.

Përgjegjja: Diagrami i klasave është dhënë në figurën vijuese.

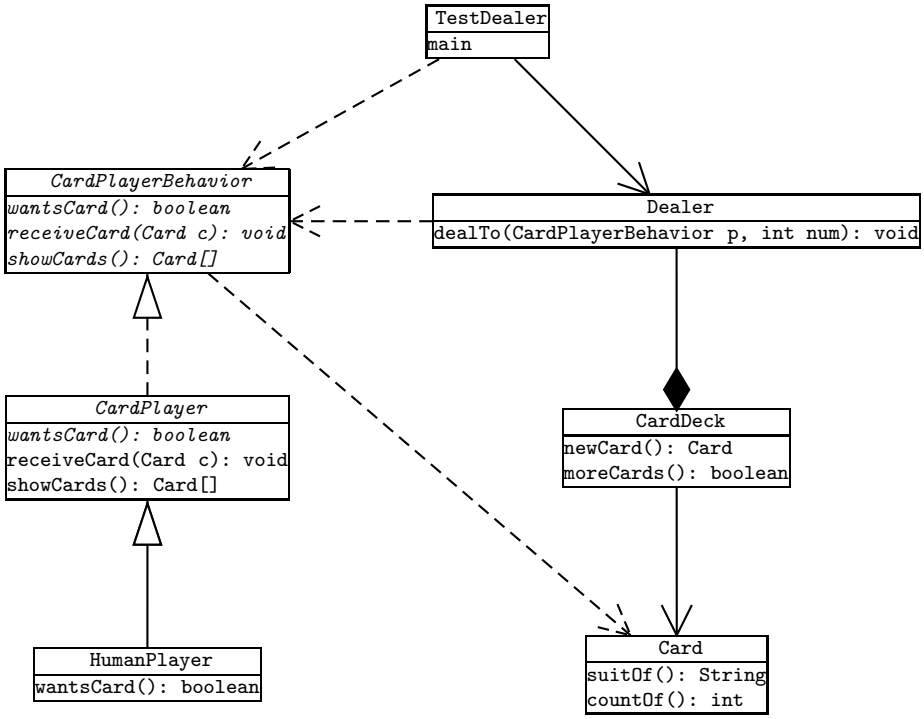

Fig. 1: Arkitektura e lojës me karta

OOP & GUI

Provimi periodik 1, Forma: B

Emri: _____

ID (Nr. dosjes): _____

Drejtimi: _____

Data: _____

Pyetjet 1–4 përbëjnë një tërësi. Për t'u përgjegjur në to, shqyrtoni klasat vijuese.

```
/** Modelon një kartë loje. */
public class Card
{ public static final String SPADES = "spades";
  public static final String HEARTS = "hearts";
  public static final String DIAMONDS = "diamonds";
  public static final String CLUBS = "clubs";
  public static final int SUIT_SIZE = 13;
  public static final int ACE = 1;
  public static final int JACK = 11;
  public static final int QUEEN = 12;
  public static final int KING = 13;

  private String suit;
  private int count;

  /** Vë vlerën dhe ngjyrën e kartës.
   *  @param s - ngjyra
   *  @param c - vlera  */
  public Card(String s, int c)
  { suit = s;
    count = c;
  }

  /** Kthen ngjyrën e kartës. */
  public String suitOf()
  { return suit; }

  /** Kthen vlerën e kartës. */
  public int countOf()
  { return count; }
}

/** Modelon një shpil kartash. */
public class CardDeck
{ private Card[] deck = new Card[4 * Card.SUIT_SIZE];
  private int count;
  /** Krijon shpil të ri me të gjitha kartat. */
  public CardDeck()
  { createSuit(Card.SPADES);
    createSuit(Card.HEARTS);
    createSuit(Card.DIAMONDS);
    createSuit(Card.CLUBS);
  }
  /** Merr një kartë të re nga shpili.
   * @return  një kartë të re (ose null nëse nuk ka karta) */
  public Card newCard()
```

```
  { Card nextCard = null;
    if ( count == 0 )
    { System.out.println("Gabim: nuk kanë mbetur karta"); }
    else
    { int index = (int)(Math.random() * count);
      nextCard = deck[index];
      for (int i = index + 1; i != count; i = i + 1)
      // kartat nga index+1...i-1 janë lëvizur në të majtë
      { deck[i-1] = deck[i]; }
      count = count - 1;
    }
    return nextCard;
  }
  /** Tregon se a ka akoma karta në shpil.
   * @return  se a është shpili jo i zbrazët */
  public boolean moreCards()
  { return count > 0; }

  private void createSuit(String s)
  { for (int i = 1; i <= Card.SUIT_SIZE; i = i + 1)
    { deck[count] = new Card(s, i);
      count = count + 1;
    }
  }
}

/** 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();
}

/** 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 që mund t'i mbajë */
  public CardPlayer(int maxCards)
  { hand = new Card[maxCards];
    count = 0;
  }

  public abstract boolean wantsCard();

  public void receiveCard(Card c)
  { hand[count] = c;
    count = count + 1;
  }
```

```
  public Card[] showCards()
  { Card[] answer = new Card[count];
    for (int i = 0; i != count; i = i + 1)
    { answer[i] = hand[i]; }
    return answer;
  }
}

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");
  }
}
```

1. Shkruani `class Dealer` sipas specifikimit të dhënë në tabelën:

`class Dealer`	modelon një shpërndarës kartash
Metodat:	
`dealTo(CardPlayerBehavior p, int n)`	lojtarit `p` i jep `numCards` karta, një nga një
Kolaboratorët:	`CardPlayerBehaviour`, `CardDeck`

Tab. 3: Specifikimi i interfejsit për shpërndarësin e kartave

2. Shkruani një aplikacion, `TestDealer`, i cili krijon 4 lojtarë dhe një objekt shpërndarës kartash, i cili pastaj secilit nga lojtarët i ndan nga 5 karta.

3. Modifikoni aplikacionin nga detyra paraprake ashtu që në dritaren komanduese të afishohet dora (kartat të cilat mban) e secilit nga lojtarët.

4. Vizatoni diagramin e klasave për aplikacionin nga detyra paraprake.

5. Është dhënë interfejsi

   ```
   public interface Convertable
   { public int double2int(double d); }
   ```

   A e implementon në mënyrë korrekte këtë interfejs klasa vijuese? Arsyetoni përgjegjjen.

   ```
   public class C2 implements Convertable
   { private double x;
     public C2(double a)
     { x = a; }
     public double double2int(double y)
     { return y; }
   }
   ```

6. Vëni konvertimet eksplicite (casting) aty ku ka nevojë ashtu që vargu vijues i instruksioneve të kompilohet në mënyrë korrekte.

```
I x = new E();
x.g();
E y = x;
y.g();
D z = new D();
z.g();
```

Çelësi i provimit B

Pyetjet 1–4 përbëjnë një tërësi. Për t'u përgjegjur në to, shqyrtoni klasat vijuese.

```
/** Modelon një kartë loje. */
public class Card
{ public static final String SPADES = "spades";
  public static final String HEARTS = "hearts";
  public static final String DIAMONDS = "diamonds";
  public static final String CLUBS = "clubs";
  public static final int SUIT_SIZE = 13;
  public static final int ACE = 1;
  public static final int JACK = 11;
  public static final int QUEEN = 12;
  public static final int KING = 13;

  private String suit;
  private int count;

  /** Vë vlerën dhe ngjyrën e kartës.
   *  @param s - ngjyra
   *  @param c - vlera  */
  public Card(String s, int c)
  { suit = s;
    count = c;
  }

  /** Kthen ngjyrën e kartës. */
  public String suitOf()
  { return suit; }

  /** Kthen vlerën e kartës. */
  public int countOf()
  { return count; }
}

/** Modelon një shpil kartash. */
public class CardDeck
{ private Card[] deck = new Card[4 * Card.SUIT_SIZE];
  private int count;
  /** Krijon shpil të ri me të gjitha kartat. */
  public CardDeck()
  { createSuit(Card.SPADES);
    createSuit(Card.HEARTS);
    createSuit(Card.DIAMONDS);
    createSuit(Card.CLUBS);
  }
  /** Merr një kartë të re nga shpili.
   * @return  një kartë të re (ose null nëse nuk ka karta) */
  public Card newCard()
  { Card nextCard = null;
    if ( count == 0 )
    { System.out.println("Gabim: nuk kanë mbetur karta"); }
```

```
    else
    { int index = (int)(Math.random() * count);
      nextCard = deck[index];
      for (int i = index + 1; i != count; i = i + 1)
      // kartat nga index+1...i-1 janë lëvizur në të majtë
      { deck[i-1] = deck[i]; }
      count = count - 1;
    }
    return nextCard;
  }
  /** Tregon se a ka akoma karta në shpil.
   * @return  se a është shpili jo i zbrazët */
  public boolean moreCards()
  { return count > 0; }

  private void createSuit(String s)
  { for (int i = 1; i <= Card.SUIT_SIZE; i = i + 1)
    { deck[count] = new Card(s, i);
      count = count + 1;
    }
  }
}

/** 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();
}

/** 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 që mund t'i mbajë */
  public CardPlayer(int maxCards)
  { hand = new Card[maxCards];
    count = 0;
  }

  public abstract boolean wantsCard();

  public void receiveCard(Card c)
  { hand[count] = c;
    count = count + 1;
  }

  public Card[] showCards()
  { Card[] answer = new Card[count];
```

```
    for (int i = 0; i != count; i = i + 1)
    { answer[i] = hand[i]; }
    return answer;
  }
}

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");
  }
}
```

1. Shkruani `class Dealer` sipas specifikimit të dhënë në tabelën:

`class Dealer`	modelon një shpërndarës kartash
Metodat:	
`dealTo(CardPlayerBehavior p, int n)`	lojtarit `p` i jep `numCards` karta, një nga një
Kolaboratorët:	`CardPlayerBehaviour`, `CardDeck`

Tab. 4: Specifikimi i interfejsit për shpërndarësin e kartave

Përgjegjja:

```
/** Modelon një shpërndarës kartash. */
public class Dealer
{ private CardDeck deck;

  /** Konstrukton shpërndarësin. */
  public Dealer()
  { deck = new CardDeck(); }

  /** I jep lojtarit të dhënë një numër të dhënë kartash,
   *  një nga një.
   *  @param p - lojtari që pranon kartat
   *  @param numCards - numri i kartave të ndara
   */
  public void dealTo(CardPlayerBehavior p, int numCards)
  { boolean processing = true;
    int i = 0;
    while ( i < numCards  &&  deck.moreCards() )
    { p.receiveCard(deck.newCard());
      i++;
    }
  }
}
```

2. Shkruani një aplikacion, `TestDealer`, i cili krijon 4 lojtarë dhe një objekt shpërndarës kartash, i cili pastaj secilit nga lojtarët i ndan nga 5 karta.

   Përgjegjja:

```
public class TestDealer {
  public static void main(String[] args) {
    Dealer dealer = new Dealer();
    CardPlayerBehavior[] cp = new CardPlayerBehavior[4];
    for ( int i = 0; i < cp.length; i++ )
    { cp[i] = new ComputerPlayer(5);
      dealer.dealTo(cp[i], 5);
    }
  }
}
```

3. Modifikoni aplikacionin nga detyra paraprake ashtu që në dritaren komanduese të afishohet dora (kartat të cilat mban) e secilit nga lojtarët.

   Përgjegjja:

```
public class TestDealer {
  public static void main(String[] args) {
    // ... njësoj sikur më parë

    for ( int i = 0; i < cp.length; i++ )
    { Card[] hand = cp[i].showCards();
      System.out.println("Dora e lojtarit " + (i + 1));
      for ( int j = 0; j < hand.length; j++ )
      {
        System.out.println(hand[j].suitOf() + " "
          + hand[j].countOf());
      }
    }
  }
}
```

4. Vizatoni diagramin e klasave për aplikacionin nga detyra paraprake.

   Përgjegjja: Diagrami i klasave është dhënë në figurën vijuese.

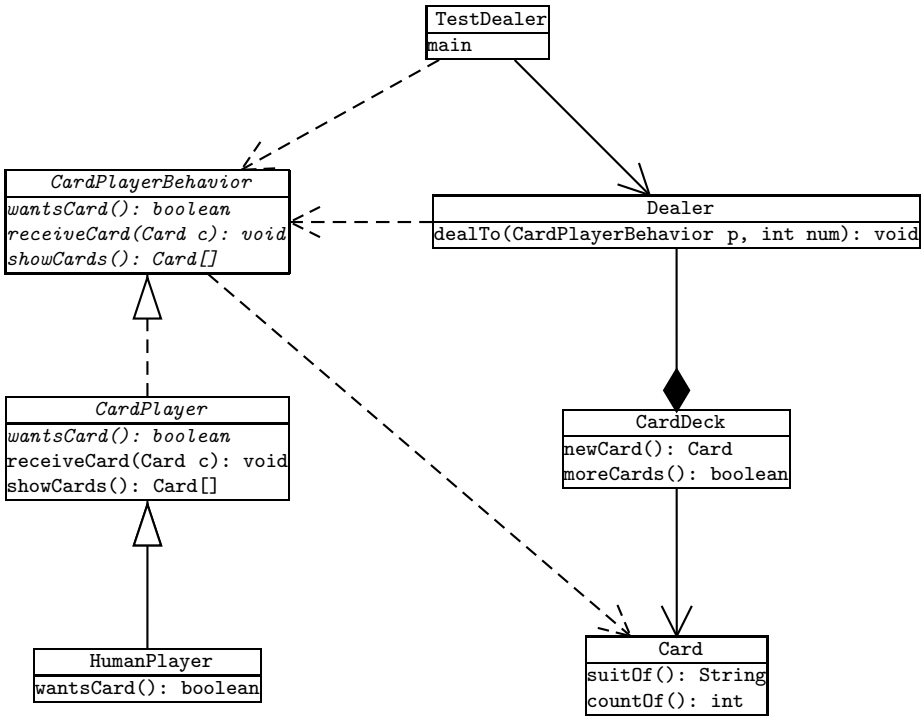

Fig. 2: Arkitektura e lojës me karta

5. Është dhënë interfejsi

```
public interface Convertable
{ public int double2int(double d); }
```

A e implementon në mënyrë korrekte këtë interfejs klasa vijuese? Arsyetoni përgjegjjen.

```
public class C2 implements Convertable
{ private double x;
  public C2(double a)
  { x = a; }
  public double double2int(double y)
  { return y; }
}
```

Përgjegjja: Jo. Metoda `double2int` e implementuar ka tipin e rezultatit të ndryshëm nga metoda e listuar në interfejsin.

6. Vëni konvertimet eksplicite (casting) aty ku ka nevojë ashtu që vargu vijues i instruksioneve të kompilohet në mënyrë korrekte.

```
I x = new E();
x.g();
E y = x;
y.g();
D z = new D();
z.g();
```

Përgjegjja:

```
I x = new E();
((E)x).g();
E y = (E)x;
y.g();
D z = new D();
z.g();
```

OOP & GUI

Provimi periodik 1, Forma: C

Emri: _____

ID (Nr. dosjes): _____

Drejtimi: _____

Data: _____

Pyetjet 1–4 përbëjnë një tërësi. Për t'u përgjegjur në to, shqyrtoni klasat vijuese.

```
/** Modelon një kartë loje. */
public class Card
{ public static final String SPADES = "spades";
  public static final String HEARTS = "hearts";
  public static final String DIAMONDS = "diamonds";
  public static final String CLUBS = "clubs";
  public static final int SUIT_SIZE = 13;
  public static final int ACE = 1;
  public static final int JACK = 11;
  public static final int QUEEN = 12;
  public static final int KING = 13;

  private String suit;
  private int count;

  /** Vë vlerën dhe ngjyrën e kartës.
   *  @param s - ngjyra
   *  @param c - vlera  */
  public Card(String s, int c)
  { suit = s;
    count = c;
  }

  /** Kthen ngjyrën e kartës. */
  public String suitOf()
  { return suit; }

  /** Kthen vlerën e kartës. */
  public int countOf()
  { return count; }
}

/** Modelon një shpil kartash. */
public class CardDeck
{ private Card[] deck = new Card[4 * Card.SUIT_SIZE];
  private int count;
  /** Krijon shpil të ri me të gjitha kartat. */
  public CardDeck()
  { createSuit(Card.SPADES);
    createSuit(Card.HEARTS);
    createSuit(Card.DIAMONDS);
    createSuit(Card.CLUBS);
  }
  /** Merr një kartë të re nga shpili.
   * @return  një kartë të re (ose null nëse nuk ka karta) */
  public Card newCard()
```

```
  { Card nextCard = null;
    if ( count == 0 )
    { System.out.println("Gabim: nuk kanë mbetur karta"); }
    else
    { int index = (int)(Math.random() * count);
      nextCard = deck[index];
      for (int i = index + 1; i != count; i = i + 1)
      // kartat nga index+1...i-1 janë lëvizur në të majtë
      { deck[i-1] = deck[i]; }
      count = count - 1;
    }
    return nextCard;
  }
  /** Tregon se a ka akoma karta në shpil.
   * @return  se a është shpili jo i zbrazët */
  public boolean moreCards()
  { return count > 0; }

  private void createSuit(String s)
  { for (int i = 1; i <= Card.SUIT_SIZE; i = i + 1)
    { deck[count] = new Card(s, i);
      count = count + 1;
    }
  }
}

/** 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();
}

/** 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 që mund t'i mbajë */
  public CardPlayer(int maxCards)
  { hand = new Card[maxCards];
    count = 0;
  }

  public abstract boolean wantsCard();

  public void receiveCard(Card c)
  { hand[count] = c;
    count = count + 1;
  }
```

```
  public Card[] showCards()
  { Card[] answer = new Card[count];
    for (int i = 0; i != count; i = i + 1)
    { answer[i] = hand[i]; }
    return answer;
  }
}

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");
  }
}
```

1. Shkruani `class Dealer` sipas specifikimit të dhënë në tabelën:

`class Dealer`	modelon një shpërndarës kartash
Metodat:	
`dealTo(CardPlayerBehavior p, int n)`	lojtarit `p` i jep `numCards` karta, një nga një
Kolaboratorët:	`CardPlayerBehaviour`, `CardDeck`

Tab. 5: Specifikimi i interfejsit për shpërndarësin e kartave

2. Shkruani një aplikacion, `TestDealer`, i cili krijon 4 lojtarë dhe një objekt shpërndarës kartash, i cili pastaj secilit nga lojtarët i ndan nga 5 karta.

3. Modifikoni aplikacionin nga detyra paraprake ashtu që në dritaren komanduese të afishohet dora (kartat të cilat mban) e secilit nga lojtarët.

4. Vizatoni diagramin e klasave për aplikacionin nga detyra paraprake.

5. Është dhënë interfejsi

   ```
   public interface Convertable
   { public int double2int(double d); }
   ```

   A e implementon në mënyrë korrekte këtë interfejs klasa vijuese? Arsyetoni përgjegjjen.

   ```
   public class C3 implements Convertable
   { private double x;
     public C3(double a)
     { x = a; }
     public int double2int(double y)
     { return (int)x; }
   }
   ```

6. Vëni konvertimet eksplicite (casting) aty ku ka nevojë ashtu që vargu vijues i instruksioneve të kompilohet në mënyrë korrekte.

```
I x = new E();
x.f(6);
if ( x instanceof E )
{ x.g(); }
E y = x;
y.g();
```

Çelësi i provimit C

Pyetjet 1–4 përbëjnë një tërësi. Për t'u përgjegjur në to, shqyrtoni klasat vijuese.

```
/** Modelon një kartë loje. */
public class Card
{ public static final String SPADES = "spades";
  public static final String HEARTS = "hearts";
  public static final String DIAMONDS = "diamonds";
  public static final String CLUBS = "clubs";
  public static final int SUIT_SIZE = 13;
  public static final int ACE = 1;
  public static final int JACK = 11;
  public static final int QUEEN = 12;
  public static final int KING = 13;

  private String suit;
  private int count;

  /** Vë vlerën dhe ngjyrën e kartës.
   *  @param s - ngjyra
   *  @param c - vlera  */
  public Card(String s, int c)
  { suit = s;
    count = c;
  }

  /** Kthen ngjyrën e kartës. */
  public String suitOf()
  { return suit; }

  /** Kthen vlerën e kartës. */
  public int countOf()
  { return count; }
}

/** Modelon një shpil kartash. */
public class CardDeck
{ private Card[] deck = new Card[4 * Card.SUIT_SIZE];
  private int count;
  /** Krijon shpil të ri me të gjitha kartat. */
  public CardDeck()
  { createSuit(Card.SPADES);
    createSuit(Card.HEARTS);
    createSuit(Card.DIAMONDS);
    createSuit(Card.CLUBS);
  }
  /** Merr një kartë të re nga shpili.
   * @return  një kartë të re (ose null nëse nuk ka karta) */
  public Card newCard()
  { Card nextCard = null;
    if ( count == 0 )
    { System.out.println("Gabim: nuk kanë mbetur karta"); }
```

```
    else
    { int index = (int)(Math.random() * count);
      nextCard = deck[index];
      for (int i = index + 1; i != count; i = i + 1)
      // kartat nga index+1...i-1 janë lëvizur në të majtë
      { deck[i-1] = deck[i]; }
      count = count - 1;
    }
    return nextCard;
  }
  /** Tregon se a ka akoma karta në shpil.
   * @return  se a është shpili jo i zbrazët */
  public boolean moreCards()
  { return count > 0; }

  private void createSuit(String s)
  { for (int i = 1; i <= Card.SUIT_SIZE; i = i + 1)
    { deck[count] = new Card(s, i);
      count = count + 1;
    }
  }
}

/** 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();
}

/** 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 që mund t'i mbajë */
  public CardPlayer(int maxCards)
  { hand = new Card[maxCards];
    count = 0;
  }

  public abstract boolean wantsCard();

  public void receiveCard(Card c)
  { hand[count] = c;
    count = count + 1;
  }

  public Card[] showCards()
  { Card[] answer = new Card[count];
```

```
    for (int i = 0; i != count; i = i + 1)
    { answer[i] = hand[i]; }
    return answer;
  }
}

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");
  }
}
```

1. Shkruani `class Dealer` sipas specifikimit të dhënë në tabelën:

`class Dealer`	modelon një shpërndarës kartash
Metodat:	
`dealTo(CardPlayerBehavior p, int n)`	lojtarit `p` i jep `numCards` karta, një nga një
Kolaboratorët:	`CardPlayerBehaviour`, `CardDeck`

Tab. 6: Specifikimi i interfejsit për shpërndarësin e kartave

Përgjegjja:

```
/** Modelon një shpërndarës kartash. */
public class Dealer
{ private CardDeck deck;

  /** Konstrukton shpërndarësin. */
  public Dealer()
  { deck = new CardDeck(); }

  /** I jep lojtarit të dhënë një numër të dhënë kartash,
   *  një nga një.
   *  @param p - lojtari që pranon kartat
   *  @param numCards - numri i kartave të ndara
   */
  public void dealTo(CardPlayerBehavior p, int numCards)
  { boolean processing = true;
    int i = 0;
    while ( i < numCards  &&  deck.moreCards() )
    { p.receiveCard(deck.newCard());
      i++;
    }
  }
}
```

2. Shkruani një aplikacion, `TestDealer`, i cili krijon 4 lojtarë dhe një objekt shpërndarës kartash, i cili pastaj secilit nga lojtarët i ndan nga 5 karta.

   Përgjegjja:

```
public class TestDealer {
  public static void main(String[] args) {
    Dealer dealer = new Dealer();
    CardPlayerBehavior[] cp = new CardPlayerBehavior[4];
    for ( int i = 0; i < cp.length; i++ )
    { cp[i] = new ComputerPlayer(5);
      dealer.dealTo(cp[i], 5);
    }
  }
}
```

3. Modifikoni aplikacionin nga detyra paraprake ashtu që në dritaren komanduese të afishohet dora (kartat të cilat mban) e secilit nga lojtarët.

   Përgjegjja:

```
public class TestDealer {
  public static void main(String[] args) {
    // ... njësoj sikur më parë

    for ( int i = 0; i < cp.length; i++ )
    { Card[] hand = cp[i].showCards();
      System.out.println("Dora e lojtarit " + (i + 1));
      for ( int j = 0; j < hand.length; j++ )
      {
        System.out.println(hand[j].suitOf() + " "
          + hand[j].countOf());
      }
    }
  }
}
```

4. Vizatoni diagramin e klasave për aplikacionin nga detyra paraprake.

   Përgjegjja: Diagrami i klasave është dhënë në figurën vijuese.

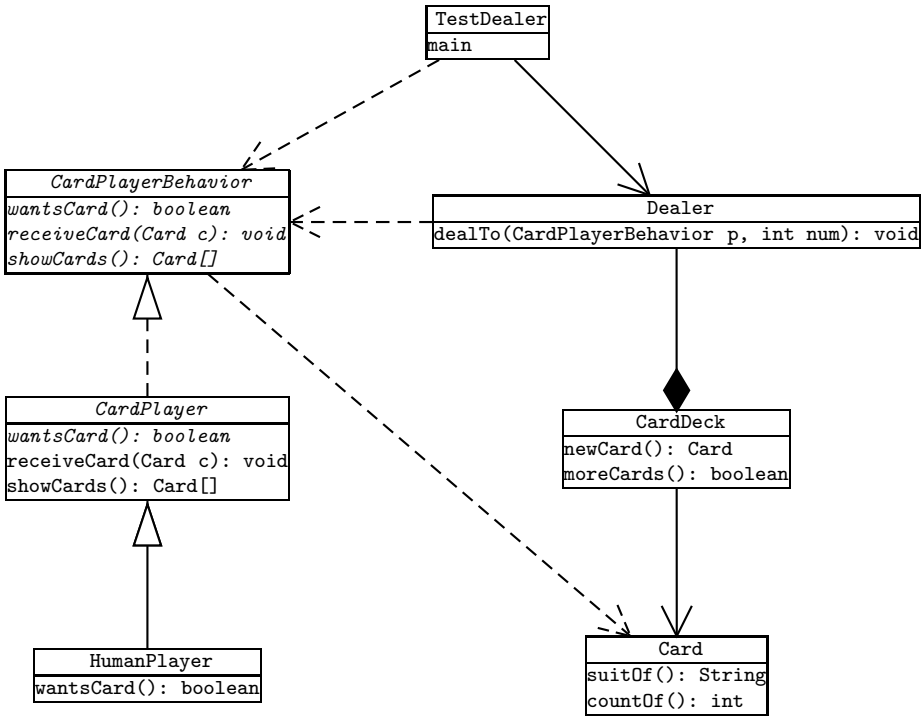

Fig. 3: Arkitektura e lojës me karta

5. Është dhënë interfejsi

```
public interface Convertable
{ public int double2int(double d); }
```

A e implementon në mënyrë korrekte këtë interfejs klasa vijuese? Arsyetoni përgjegjjen.

```
public class C3 implements Convertable
{ private double x;
  public C3(double a)
  { x = a; }
  public int double2int(double y)
  { return (int)x; }
}
```

Përgjegjja: Po. Është implementuar metoda e listuar në interfejsin.

6. Vëni konvertimet eksplicite (casting) aty ku ka nevojë ashtu që vargu vijues i instruksioneve të kompilohet në mënyrë korrekte.

```
I x = new E();
x.f(6);
if ( x instanceof E )
{ x.g(); }
E y = x;
y.g();
```

Përgjegjja:

```
I x = new E();
x.f(6);
if ( x instanceof E )
{ ((E)x).g(); }
E y = (E)x;
y.g();
```

OOP & GUI

Provimi periodik 1, Forma: D

Emri: _____

ID (Nr. dosjes): _____

Drejtimi: _____

Data: _____

1. Është dhënë interfejsi

   ```
   public interface Convertable
   { public int double2int(double d); }
   ```

   A e implementon në mënyrë korrekte këtë interfejs klasa vijuese? Arsyetoni përgjegjjen.

   ```
   public class C1 implements Convertable
   { private double x;
     public C1(double a)
     { x = a; }
     public int double2int(double y)
     { return (int)(x + y); }
   }
   ```

Pyetjet 2–5 përbëjnë një tërësi. Për t'u përgjegjur në to, shqyrtoni klasat vijuese.

```
/** Modelon një kartë loje. */
public class Card
{ public static final String SPADES = "spades";
  public static final String HEARTS = "hearts";
  public static final String DIAMONDS = "diamonds";
  public static final String CLUBS = "clubs";
  public static final int SUIT_SIZE = 13;
  public static final int ACE = 1;
  public static final int JACK = 11;
  public static final int QUEEN = 12;
  public static final int KING = 13;

  private String suit;
  private int count;

  /** Vë vlerën dhe ngjyrën e kartës.
   *  @param s - ngjyra
   *  @param c - vlera  */
  public Card(String s, int c)
  { suit = s;
    count = c;
  }

  /** Kthen ngjyrën e kartës. */
  public String suitOf()
  { return suit; }

  /** Kthen vlerën e kartës. */
  public int countOf()
  { return count; }
}
```

```
/** Modelon një shpil kartash. */
public class CardDeck
{ private Card[] deck = new Card[4 * Card.SUIT_SIZE];
  private int count;
  /** Krijon shpil të ri me të gjitha kartat. */
  public CardDeck()
  { createSuit(Card.SPADES);
    createSuit(Card.HEARTS);
    createSuit(Card.DIAMONDS);
    createSuit(Card.CLUBS);
  }
  /** Merr një kartë të re nga shpili.
   * @return  një kartë të re (ose null nëse nuk ka karta) */
  public Card newCard()
  { Card nextCard = null;
    if ( count == 0 )
    { System.out.println("Gabim: nuk kanë mbetur karta"); }
    else
    { int index = (int)(Math.random() * count);
      nextCard = deck[index];
      for (int i = index + 1; i != count; i = i + 1)
      // kartat nga index+1...i-1 janë lëvizur në të majtë
      { deck[i-1] = deck[i]; }
      count = count - 1;
    }
    return nextCard;
  }
  /** Tregon se a ka akoma karta në shpil.
   * @return  se a është shpili jo i zbrazët */
  public boolean moreCards()
  { return count > 0; }

  private void createSuit(String s)
  { for (int i = 1; i <= Card.SUIT_SIZE; i = i + 1)
    { deck[count] = new Card(s, i);
      count = count + 1;
    }
  }
}

/** 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();
}

/** 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 që mund t'i mbajë */
  public CardPlayer(int maxCards)
  { hand = new Card[maxCards];
    count = 0;
  }

  public abstract boolean wantsCard();

  public void receiveCard(Card c)
  { hand[count] = c;
    count = count + 1;
  }

  public Card[] showCards()
  { Card[] answer = new Card[count];
    for (int i = 0; i != count; i = i + 1)
    { answer[i] = hand[i]; }
    return answer;
  }
}
```

```
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");
  }
}
```

2. Shkruani `class Dealer` sipas specifikimit të dhënë në tabelën:

`class Dealer`	modelon një shpërndarës kartash
Metodat:	
`dealTo(CardPlayerBehavior p, int n)`	lojtarit `p` i jep `numCards` karta, një nga një
Kolaboratorët:	`CardPlayerBehaviour`, `CardDeck`

Tab. 7: Specifikimi i interfejsit për shpërndarësin e kartave

3. Shkruani një aplikacion, `TestDealer`, i cili krijon 4 lojtarë dhe një objekt shpërndarës kartash, i cili pastaj secilit nga lojtarët i ndan nga 5 karta.

4. Modifikoni aplikacionin nga detyra paraprake ashtu që në dritaren komanduese të afishohet dora (kartat të cilat mban) e secilit nga lojtarët.

5. Vizatoni diagramin e klasave për aplikacionin nga detyra paraprake.

6. Vëni konvertimet eksplicite (casting) aty ku ka nevojë ashtu që vargu vijues i instruksioneve të kompilohet në mënyrë korrekte.

```
I x = new E();
x.g();
E y = x;
y.g();
D z = new D();
z.g();
```

Çelësi i provimit D

1. Është dhënë interfejsi

   ```
   public interface Convertable
   { public int double2int(double d); }
   ```

   A e implementon në mënyrë korrekte këtë interfejs klasa vijuese? Arsyetoni përgjegjjen.

   ```
   public class C1 implements Convertable
   { private double x;
     public C1(double a)
     { x = a; }
     public int double2int(double y)
     { return (int)(x + y); }
   }
   ```

   Përgjegjja: Po. Është implementuar metoda e listuar në interfejsin.

Pyetjet 2–5 përbëjnë një tërësi. Për t'u përgjegjur në to, shqyrtoni klasat vijuese.

```
/** Modelon një kartë loje. */
public class Card
{ public static final String SPADES = "spades";
  public static final String HEARTS = "hearts";
  public static final String DIAMONDS = "diamonds";
  public static final String CLUBS = "clubs";
  public static final int SUIT_SIZE = 13;
  public static final int ACE = 1;
  public static final int JACK = 11;
  public static final int QUEEN = 12;
  public static final int KING = 13;

  private String suit;
  private int count;

  /** Vë vlerën dhe ngjyrën e kartës.
   *  @param s - ngjyra
   *  @param c - vlera  */
  public Card(String s, int c)
  { suit = s;
    count = c;
  }

  /** Kthen ngjyrën e kartës. */
  public String suitOf()
  { return suit; }

  /** Kthen vlerën e kartës. */
  public int countOf()
  { return count; }
}
```

```
/** Modelon një shpil kartash. */
public class CardDeck
{ private Card[] deck = new Card[4 * Card.SUIT_SIZE];
  private int count;
  /** Krijon shpil të ri me të gjitha kartat. */
  public CardDeck()
  { createSuit(Card.SPADES);
    createSuit(Card.HEARTS);
    createSuit(Card.DIAMONDS);
    createSuit(Card.CLUBS);
  }
  /** Merr një kartë të re nga shpili.
   * @return  një kartë të re (ose null nëse nuk ka karta) */
  public Card newCard()
  { Card nextCard = null;
    if ( count == 0 )
    { System.out.println("Gabim: nuk kanë mbetur karta"); }
    else
    { int index = (int)(Math.random() * count);
      nextCard = deck[index];
      for (int i = index + 1; i != count; i = i + 1)
      // kartat nga index+1...i-1 janë lëvizur në të majtë
      { deck[i-1] = deck[i]; }
      count = count - 1;
    }
    return nextCard;
  }
  /** Tregon se a ka akoma karta në shpil.
   * @return  se a është shpili jo i zbrazët */
  public boolean moreCards()
  { return count > 0; }

  private void createSuit(String s)
  { for (int i = 1; i <= Card.SUIT_SIZE; i = i + 1)
    { deck[count] = new Card(s, i);
      count = count + 1;
    }
  }
}

/** 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();
}

/** 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 që mund t'i mbajë */
  public CardPlayer(int maxCards)
  { hand = new Card[maxCards];
    count = 0;
  }

  public abstract boolean wantsCard();

  public void receiveCard(Card c)
  { hand[count] = c;
    count = count + 1;
  }

  public Card[] showCards()
  { Card[] answer = new Card[count];
    for (int i = 0; i != count; i = i + 1)
    { answer[i] = hand[i]; }
    return answer;
  }
}
```

```
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");
  }
}
```

2. Shkruani `class Dealer` sipas specifikimit të dhënë në tabelën:

`class Dealer`	modelon një shpërndarës kartash
Metodat:	
`dealTo(CardPlayerBehavior p, int n)`	lojtarit `p` i jep `numCards` karta, një nga një
Kolaboratorët:	`CardPlayerBehaviour`, `CardDeck`

Tab. 8: Specifikimi i interfejsit për shpërndarësin e kartave

Përgjegjja:

```
/** Modelon një shpërndarës kartash. */
public class Dealer
{ private CardDeck deck;

  /** Konstrukton shpërndarësin. */
  public Dealer()
```

```
  { deck = new CardDeck(); }

  /** I jep lojtarit të dhënë një numër të dhënë kartash,
   *  një nga një.
   *  @param p - lojtari që pranon kartat
   *  @param numCards - numri i kartave të ndara
   */
  public void dealTo(CardPlayerBehavior p, int numCards)
  { boolean processing = true;
    int i = 0;
    while ( i < numCards  &&  deck.moreCards() )
    { p.receiveCard(deck.newCard());
      i++;
    }
  }
}
```

3. Shkruani një aplikacion, `TestDealer`, i cili krijon 4 lojtarë dhe një objekt shpërndarës kartash, i cili pastaj secilit nga lojtarët i ndan nga 5 karta.

   Përgjegjja:

```
public class TestDealer {
  public static void main(String[] args) {
    Dealer dealer = new Dealer();
    CardPlayerBehavior[] cp = new CardPlayerBehavior[4];
    for ( int i = 0; i < cp.length; i++ )
    { cp[i] = new ComputerPlayer(5);
      dealer.dealTo(cp[i], 5);
    }
  }
}
```

4. Modifikoni aplikacionin nga detyra paraprake ashtu që në dritaren komanduese të afishohet dora (kartat të cilat mban) e secilit nga lojtarët.

   Përgjegjja:

```
public class TestDealer {
  public static void main(String[] args) {
    // ... njësoj sikur më parë

    for ( int i = 0; i < cp.length; i++ )
    { Card[] hand = cp[i].showCards();
      System.out.println("Dora e lojtarit " + (i + 1));
      for ( int j = 0; j < hand.length; j++ )
      {
        System.out.println(hand[j].suitOf() + " "
          + hand[j].countOf());
      }
    }
  }
}
```

5. Vizatoni diagramin e klasave për aplikacionin nga detyra paraprake.

Përgjegjja: Diagrami i klasave është dhënë në figurën vijuese.

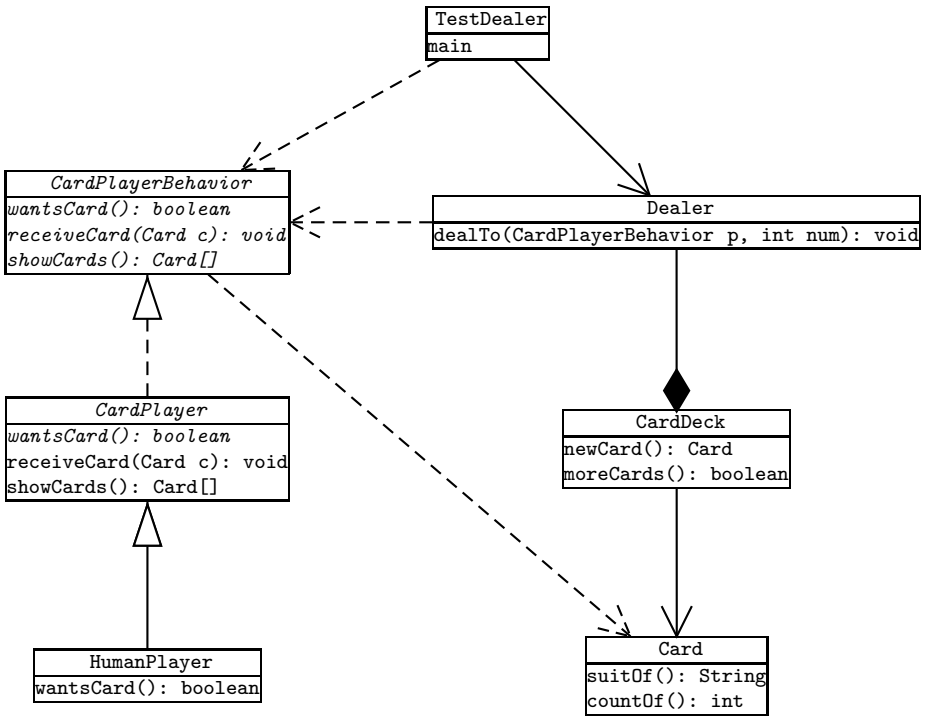

Fig. 4: Arkitektura e lojës me karta

6. Vëni konvertimet eksplicite (casting) aty ku ka nevojë ashtu që vargu vijues i instruksioneve të kompilohet në mënyrë korrekte.

```
I x = new E();
x.g();
E y = x;
y.g();
D z = new D();
z.g();
```

Përgjegjja:

```
I x = new E();
((E)x).g();
E y = (E)x;
y.g();
D z = new D();
z.g();
```