

Blackjack dokumentációja

Tartalomjegyzék

1. [Osztálydiagram](#)
2. [Card osztály](#)
3. [Deck osztály](#)
4. [Rank osztály](#)
5. [Suit osztály](#)
6. [Hand osztály](#)
7. [CoinPurse osztály](#)
8. [Dealer osztály](#)
9. [GameUI osztály](#)
10. [Main osztály](#)
11. [Felhasználói dokumentáció](#)

Osztálydiagram



Card

Egy kártya, amely három attribútummal rendelkezik: rang (**rank**), szín (**suit**), és a kártya képének elérési útja (**image**).

Konstruktor

Card(Rank r, Suit s, Image img)

Létrehoz egy új **Card** objektumot a megadott paraméterekkel.

- **Paraméterek:**

- **r (Rank)**: A kártya rangja (pl. **ACE**, **TWO**, **THREE**, ..., **KING**).
- **s (Suit)**: A kártya színe (pl. **CLUBS**, **DIAMONDS**, **HEARTS**, **SPADES**).
- **img (Image)**: A kártya képének fájl elérési útja.

- **Példa:**

```
Card card = new Card(Rank.ACE, Suit.HEARTS, new  
ImageIcon("path/to/image.png").getImage());
```

Metódusok

String getSuit()

Visszaadja a kártya színét.

- **Visszatérési érték:**

- **String**: A kártya színe.

- **Példa:**

```
String suit = card.getSuit();
```

int getRank()

Visszaadja a kártya rangját.

- **Visszatérési érték:**

- **int**: A kártya rangja.

- **Példa:**

```
int rank = card.getRank();
```

Image getImage()

Visszaadja a kártya képét.

- **Visszatérési érték:**
 - **Image:** A kártya képe.

- **Példa:**

```
Image image = card.getImage();
```

String toString()

Visszaadja a kártya sztring reprezentációját a "rank_of_suit.png" formátumban.

- **Visszatérési érték:**
 - **String:** A kártya sztring reprezentációja.

- **Példa:**

```
String cardString = card.toString();
```

Deck osztály

Áttekintés

A **Deck** osztály a kártyapaklit reprezentálja a Blackjack játékban. Ez az osztály tartalmazza az 52 különböző kártyát, és biztosítja a kártyák keverésére és húzására szolgáló metódusokat.

Osztály

Deck

Egy kártyapakli, amely két attribútummal rendelkezik: a kártyák tárolója (**cardStorage**) és a megkevert pakli (**shuffledDeck**).

Konstruktor

Deck()

Létrehoz egy új **Deck** objektumot, amely tárolja az 52 különböző kártyát sorrendben.

- **Példa:**

```
Deck deck = new Deck();
```

Metódusok

`void shuffle()`

A kártyatárolót használja egy új, 52 kártyából álló pakli megkeveréséhez.

- **Példa:**

```
deck.shuffle();
```

`Card drawCard()`

Visszaadja a pakli első kártyáját. Ha a pakli üres (minden kártyát kihúztak vagy a `Deck` objektumot éppen most hozták létre), a `shuffle()` metódust használja a feltöltéshez.

- **Visszatérési érték:**

- `Card`: A pakli első kártyája.

- **Példa:**

```
Card card = deck.drawCard();
```

Példa használat

```
public class Main {  
    public static void main(String[] args) {  
        Deck deck = new Deck(); // Új pakli létrehozása  
        deck.shuffle(); // Pakli megkeverése  
        Card card = deck.drawCard(); // Kártya húzása  
        System.out.println("Húzott kártya: " + card);  
    }  
}
```

Rank osztály

Áttekintés

A `Rank` osztály egy felsoroló (enum), amely a Blackjack játékban használt alapvető játékkártyák rangjait reprezentálja. Ez az osztály tartalmazza a rangok nevét, szimbólumát és értékét, amelyeket a Blackjack játékban használnak.

Osztály

Rank

Egy felsoroló, amely a kártyák rangjait tartalmazza. Minden ranghoz tartozik egy szimbólum (**symbol**) és egy érték (**value**).

Konstruktor

Rank(String sym, int value)

Létrehoz egy új **Rank** objektumot a megadott szimbólummal és értékkel.

- **Paraméterek:**
 - **sym** (**String**): A rang szimbóluma (pl. **A**, **K**, **Q**, **J**, **10**, stb.).
 - **value** (**int**): A rang értéke (pl. **1**, **10**, **9**, stb.).

- **Példa:**

```
Rank ace = Rank.ACE;
```

Attribútumok

String symbol

A rang szimbóluma (pl. **A**, **K**, **Q**, **J**, **10**, stb.).

int value

A rang értéke (pl. **1**, **10**, **9**, stb.).

Példa használat

```
public class Main {  
    public static void main(String[] args) {  
        Rank rank = Rank.ACE;  
        System.out.println("Rang: " + rank);  
        System.out.println("Szimbólum: " + rank.symbol);  
        System.out.println("Érték: " + rank.value);  
    }  
}
```

Suit osztály

Áttekintés

A **Suit** osztály egy felsoroló (enum), amely a Blackjack játékban használt alapvető játékkártyák színeit reprezentálja. Ez az osztály tartalmazza a színek szimbólumát és betűjelét, amelyeket a kártyák grafikus megjelenítéséhez és kódolásához használnak.

Osztály

Suit

Egy felsoroló, amely a kártyák színeit tartalmazza. Minden színhez tartozik egy szimbólum (**symbol**) és egy betűjel (**letter**).

Konstruktor

Suit(char symbol, char letter)

Létrehoz egy új **Suit** objektumot a megadott szimbólummal és betűjellel.

- **Paraméterek:**

- **symbol (char)**: A kártyaszín szimbóluma (pl. ♣, ♠, ♥, ♦).
- **letter (char)**: A kártyaszín betűjele (pl. c, s, h, d).

- **Példa:**

```
Suit clubs = Suit.CLUBS;
```

Attribútumok

char symbol

A kártyaszín szimbóluma.

char letter

A kártyaszín betűjele.

Példa használat

```
public class Main {  
    public static void main(String[] args) {  
        Suit suit = Suit.HEARTS;  
        System.out.println("Szín: " + suit);  
        System.out.println("Szimbólum: " + suit.symbol);  
        System.out.println("Betűjel: " + suit.letter);  
    }  
}
```

Hand osztály

Áttekintés

A **Hand** osztály egyetlen kezet reprezentál a Blackjack játékban. Ez az osztály tartalmazza a kártyák listáját, a kéz értékét, a tét összegét és a kéz tulajdonosát. Biztosítja a kártyák hozzáadására, a kéz értékének lekérdezésére és a kéz osztására szolgáló metódusokat.

Osztály

Hand

Egy kéz, amely több attribútummal rendelkezik: kártyák listája (**cards**), a kéz értéke (**value**), az ász értéke (**acedValue**), a tét összege (**bet**), az ász jelenléte (**hasAce**), az osztás állapota (**isSplitted**) és a kéz tulajdonosa (**owner**).

Konstruktor

Hand(String owner)

Létrehoz egy új **Hand** objektumot és megadja a tulajdonost.

- **Paraméterek:**
 - **owner** (**String**): A kéz tulajdonosa (pl. **player**, **dealer**).
- **Példa:**

```
Hand hand = new Hand("player");
```

Metódusok

void addCard(Card c)

Hozzáad egy adott kártyát a kézhez. Ha a kártya null, visszatér anélkül, hogy bármit is tenne.

- **Paraméterek:**
 - **c** (**Card**): A kézhez osztott kártya.
- **Példa:**

```
hand.addCard(new Card(Rank.ACE, Suit.CLUBS, null));
```

boolean blackJack()

Ellenőrzi, hogy a kéz tartalmaz-e BlackJack-et (21 érték az első két kártyával, ÁSZ és 10 vagy J vagy Q vagy K).

- **Visszatérési érték:**

- **boolean**: true, ha a kéz BlackJack-et tartalmaz.

- **Példa:**

```
boolean isBlackJack = hand.blackJack();
```

boolean isSplittable()

Ellenőrzi, hogy a kéz osztható-e. A kéz osztható, ha a tulajdonosa a játékos, a kéz két kártyát tartalmaz, és a két kártya értéke megegyezik.

- **Visszatérési érték:**

- **boolean**: true, ha a kéz osztható.

- **Példa:**

```
boolean canSplit = hand.isSplittable();
```

Hand split()

Létrehoz egy új kezet az eredeti kéz egyik kártyájának felhasználásával. Ha a kéz nem osztható, null értéket ad vissza.

- **Visszatérési érték:**

- **Hand**: Az osztott kéz vagy null, ha a kéz nem osztható.

- **Példa:**

```
Hand splitHand = hand.split();
```

int getAcedValue()

Visszaadja a kéz értékét, amikor az Ász 11-nek számít. Ha az érték meghaladja a 21-et, 0-t ad vissza.

- **Visszatérési érték:**

- **int**: A kéz értéke, amikor az Ász 11-nek számít.

- **Példa:**

```
int acedValue = hand.getAcedValue();
```

void addBet(int b)

Hozzáad egy adott összeget a kézre tett tét összegéhez. Ha kevesebb, mint 10, 10-et ad hozzá.

- **Paraméterek:**
 - `b (int)`: A megadott tét.

- **Példa:**

```
hand.addBet(50);
```

Card getCard(int index)

Visszaadja a megadott indexhez osztott kártyát.

- **Paraméterek:**
 - `index (int)`: A kívánt kártya indexe.
- **Visszatérési érték:**
 - `Card`: A megadott indexhez tartozó kártya.

- **Példa:**

```
Card card = hand.getCard(0);
```

int getValue()

Visszaadja a kéz értékét, az ászokat 1-nek tekintik.

- **Visszatérési érték:**
 - `int`: A kéz értéke.

- **Példa:**

```
int value = hand.getValue();
```

String getOwner()

Visszaadja a kéz tulajdonosát.

- **Visszatérési érték:**

- **String**: A tulajdonos.

- **Példa:**

```
String owner = hand.getOwner();
```

boolean hasAce()

Visszaadja, hogy a kéz tartalmaz-e Ászt.

- **Visszatérési érték:**

- **boolean**: true, ha a kéz Ászt tartalmaz.

- **Példa:**

```
boolean containsAce = hand.hasAce();
```

int getBet()

Visszaadja a kézre tett tét összegét.

- **Visszatérési érték:**

- **int**: A kézre tett tét összege.

- **Példa:**

```
int bet = hand.getBet();
```

int getSize()

Visszaadja a kéz méretét tesztelés céljából.

- **Visszatérési érték:**

- **int**: A kéz mérete.

- **Példa:**

```
int size = hand.getSize();
```

CoinPurse osztály

Áttekintés

A **CoinPurse** osztály kezeli a játékos pénzét és fogadásait a Blackjack játékban. Ez az osztály tartalmazza a játékos aktuális egyenlegét és tétjét, valamint biztosítja a pénz kezelésére és a fogadások kezelésére szolgáló metódusokat.

Osztály

CoinPurse

Egy pénztárca, amely két attribútummal rendelkezik: a játékos aktuális egyenlege (**balance**) és a játékos aktuális tétje (**currentBet**).

Konstruktor

CoinPurse(int initialBalance)

Létrehoz egy új **CoinPurse** objektumot a megadott kezdő egyenleggel. Ha a kezdő egyenleg kisebb vagy egyenlő nullával, akkor az alapértelmezett egyenleg 2000 lesz.

- **Paraméterek:**

- **initialBalance** (**int**): A kezdő egyenleg.

- **Példa:**

```
CoinPurse coinPurse = new CoinPurse(1000);
```

Metódusok

boolean betAcceptable()

Ellenőrzi, hogy a **CoinPurse** objektumnak van-e elég pénze a játékos által kiválasztott összeg fogadásához.

- **Visszatérési érték:**

- **boolean**: true, ha van elég pénz, különben false.

- **Példa:**

```
boolean canBet = coinPurse.betAcceptable();
```

int bet()

Fogadást tesz a kiválasztott összegre, és visszaadja a fogadott összeget. Ha nincs elég pénz a **CoinPurse** objektumban, nullát ad vissza.

- **Visszatérési érték:**

- `int`: A fogadott összeg.

- **Példa:**

```
int betAmount = coinPurse.bet();
```

`int getBalance()`

Visszaadja a `CoinPurse` aktuális egyenlegét.

- **Visszatérési érték:**

- `int`: Az aktuális egyenleg.

- **Példa:**

```
int balance = coinPurse.getBalance();
```

`int getCurrentBet()`

Visszaadja a `CoinPurse` aktuális tétjét.

- **Visszatérési érték:**

- `int`: Az aktuális tét.

- **Példa:**

```
int currentBet = coinPurse.getCurrentBet();
```

`void addMoney(int m)`

Hozzáadja a megadott pénzösszeget a `CoinPurse` objektumhoz. Ha negatív összeget adnak meg, nem ad hozzá semmit.

- **Paraméterek:**

- `m (int)`: A hozzáadandó pénzösszeg.

- **Példa:**

```
coinPurse.addMoney(500);
```

`void setWealth(int m)`

Beállítja a **CoinPurse** jelenlegi vagyonát a megadott összegre. Ha kevesebb, mint 10, megtartja a jelenlegi összeget.

- **Paraméterek:**

- **m** (**int**): A beállítandó vagyon összege.

- **Példa:**

```
coinPurse.setWealth(1500);
```

void setBetSize(int m)

Beállítja a **CoinPurse** jelenlegi tétméretét a megadott összegre. Ha kevesebb, mint 10, megtartja a jelenlegi összeget.

- **Paraméterek:**

- **m** (**int**): A beállítandó tétméret.

- **Példa:**

```
// A metódus privát, így közvetlenül nem hívható meg.
```

void stateChanged(ChangeEvent e)

Kezeli a **JSlider** állapotváltozását, és beállítja a tétméretet a csúszka értékére.

- **Paraméterek:**

- **e** (**ChangeEvent**): Az állapotváltozás eseménye.

- **Példa:**

```
// A metódus automatikusan meghívódik, amikor a JSlider értéke megváltozik.
```

Példa használat

```
public class Main {
    public static void main(String[] args) {
        CoinPurse coinPurse = new CoinPurse(1000);
        System.out.println("Kezdő egyenleg: " + coinPurse.getBalance());
        coinPurse.addMoney(500);
        System.out.println("Új egyenleg: " + coinPurse.getBalance());
        if (coinPurse.betAcceptable()) {
```

```
        int betAmount = coinPurse.bet();
        System.out.println("Fogadott összeg: " + betAmount);
    } else {
        System.out.println("Nincs elég pénz a fogadáshoz.");
    }
}
}
```

Dealer osztály

Áttekintés

A **Dealer** osztály a Blackjack játék logikáját és a kártyák osztását kezeli. Ez az osztály tartalmazza a paklit, a felhasználói felületet, a játékos és az osztó kezeit, valamint a játékos pénztárcáját. Biztosítja a kártyák osztására, a játék állapotának frissítésére és a játék végének kezelésére szolgáló metódusokat.

Osztály

Dealer

A játék logikáját és a kártyák osztását kezelő osztály.

Konstruktor

Dealer()

Létrehoz egy új **Dealer** objektumot, amely inicializálja a paklit, a felhasználói felületet és a játékos pénztárcáját.

- **Példa:**

```
Dealer dealer = new Dealer();
```

Metódusok

void updateHandValue(Hand hand)

Frissíti a kéz értékét a felhasználói felületen.

- **Paraméterek:**

- **hand (Hand):** A frissítendő kéz.

- **Példa:**

```
dealer.updateHandValue(playerHand);
```

void dealCard(Hand target)

Kioszt egy kártyát a megadott kézhez.

- **Paraméterek:**

(**Hand**): A cél kéz, amelyhez a kártyát ki kell osztani.

- **Példa:**

```
dealer.dealCard(playerHand);
```

boolean checkBust(Hand hand)

Ellenőrzi, hogy a kéz értéke meghaladja-e a 21-et (bust).

- **Paraméterek:**

- **hand (Hand)**: Az ellenőrizendő kéz.

- **Visszatérési érték:**

- **boolean**: true, ha a kéz értéke meghaladja a 21-et, különben false.

- **Példa:**

```
boolean isBust = dealer.checkBust(playerHand);
```

boolean gameoverBust(Hand hand)

Kezeli a játék végét, ha a kéz értéke meghaladja a 21-et.

- **Paraméterek:**

- **hand (Hand)**: Az ellenőrizendő kéz.

- **Visszatérési érték:**

- **boolean**: true, ha a játék véget ért, különben false.

- **Példa:**

```
boolean isGameOver = dealer.gameoverBust(playerHand);
```

void setEnableSplitHand(boolean enable)

Engedélyezi vagy letiltja a megosztott kéz használatát.

- **Paraméterek:**

- `enable (boolean)`: true, ha engedélyezni kell a megosztott kéz használatát, különben false.

- **Példa:**

```
dealer.setEnableSplitHand(true);
```

`void doubleHand(Hand hand)`

Megduplázza a tétet és kioszt egy kártyát a megadott kézhez.

- **Paraméterek:**

- `hand (Hand)`: A megduplázandó kéz.

- **Példa:**

```
dealer.doubleHand(playerHand);
```

`void payWinnings(int amount)`

Kifizeti a nyereményt a játékosnak.

- **Paraméterek:**

- `amount (int)`: A kifizetendő összeg.

- **Példa:**

```
dealer.payWinnings(100);
```

`void declareSplitWinnings(int dealer, int player, int split)`

Kifizeti a nyereményt a megosztott kéz esetén.

- **Paraméterek:**

- `dealer (int)`: Az osztó értéke.
- `player (int)`: A játékos értéke.
- `split (int)`: A megosztott kéz értéke.

- **Példa:**

```
dealer.declareSplitWinnings(20, 18, 19);
```

void declareWinner()

Meghatározza a győztest a játék végén.

- **Példa:**

```
dealer.declareWinner();
```

boolean checkBlackJack()

Ellenőrzi, hogy a játékos vagy az osztó kapott-e BlackJack-et.

- **Visszatérési érték:**

- **boolean**: true, ha valamelyik kapott BlackJack-et, különben false.

- **Példa:**

```
boolean hasBlackJack = dealer.checkBlackJack();
```

void dealPressed()

Kezeli a "deal" gomb megnyomását.

- **Példa:**

```
dealer.dealPressed();
```

void hitPressed()

Kezeli a "hit" gomb megnyomását.

- **Példa:**

```
dealer.hitPressed();
```

void stayPressed()

Kezeli a "stay" gomb megnyomását.

- **Példa:**

```
dealer.stayPressed();
```

void splitPressed()

Kezeli a "split" gomb megnyomását.

- **Példa:**

```
dealer.splitPressed();
```

void actionPerformed(ActionEvent e)

Kezeli a felhasználói felület eseményeit.

- **Paraméterek:**

- **e** (**ActionEvent**): Az esemény, amelyet kezelni kell.

- **Példa:**

```
dealer.actionPerformed(new ActionEvent(this,  
ActionEvent.ACTION_PERFORMED, "deal"));
```

Hand getHand()

Visszaadja a játékos kezét.

- **Visszatérési érték:**

- **Hand**: A játékos keze.

- **Példa:**

```
Hand playerHand = dealer.getHand();
```

void hit()

Végrehajt egy hit műveletet.

- **Példa:**

```
dealer.hit();
```

void surrender()

Végrehajt egy surrender műveletet. A játékos feladja a játékot, és visszakapja a tét felét.

- **Példa:**

```
dealer.surrender();
```

int getBalance()

Visszaadja a játékos egyenlegét.

- **Visszatérési érték:**

- `int`: A játékos egyenlege.

- **Példa:**

```
int balance = dealer.getBalance();
```

void deal()

Végrehajt egy deal műveletet.

- **Példa:**

```
dealer.deal();
```

Példa használat

```
public class Main {  
    public static void main(String[] args) {  
        Dealer dealer = new Dealer();  
        dealer.deal();  
        dealer.hit();  
        dealer.stayPressed();  
        System.out.println("Játékos egyenlege: " + dealer.getBalance());  
    }  
}
```

GameUI osztály

Áttekintés

A **GameUI** osztály a Blackjack játék grafikus felhasználói felületét (GUI) kezeli. Ez az osztály tartalmazza a játék különböző elemeit, mint például a gombokat, csúszkákat, kártyákat és a játék állapotát jelző címkéket. Biztosítja a felhasználói interakciók kezelésére szolgáló metódusokat.

Osztály

GameUI

A játék grafikus felhasználói felületét kezelő osztály.

Konstruktor

GameUI(ActionListener list)

Létrehoz egy új **GameUI** objektumot, amely inicializálja a játék felhasználói felületét és beállítja a gombokhoz tartozó eseménykezelőket.

- **Paraméterek:**

- **list** (**ActionListener**): Az eseménykezelő, amely kezeli a gombnyomásokat.

- **Példa:**

```
GameUI gameUI = new GameUI(new MyActionListener());
```

Metódusok

void drawCard(Card c, String dest)

Kártya kirajzolása a megadott célhelyre.

- **Paraméterek:**

- **c** (**Card**): A kirajzolandó kártya.
- **dest** (**String**): A célhely (pl. **dealer**, **player**, **split**).

- **Példa:**

```
gameUI.drawCard(new Card(Rank.ACE, Suit.HEARTS, "/path/to/image.png"),  
"player");
```

void setEnableButton(String command)

Gombok engedélyezése vagy letiltása a megadott parancs alapján.

- **Paraméterek:**

- **command** (*String*): A parancs, amely alapján a gombokat engedélyezni vagy letiltani kell (pl. *deal*, *stay*, *hit*, *double*, *split*).

- **Példa:**

```
gameUI.setEnabledButton("deal");
```

void updateHandValue(String owner, String value)

Frissíti a kéz értékét a felhasználói felületen.

- **Paraméterek:**

- **owner** (*String*): A kéz tulajdonosa (pl. *dealer*, *player*, *split*).
- **value** (*String*): A kéz értéke.

- **Példa:**

```
gameUI.updateHandValue("player", "21");
```

void clear()

Törli a kártyákat a felhasználói felületről.

- **Példa:**

```
gameUI.clear();
```

void clearPlayer()

Törli a játékos kártyáit a felhasználói felületről.

- **Példa:**

```
gameUI.clearPlayer();
```

void disableHand(String owner)

Letiltja a kéz kártyáit a felhasználói felületen.

- **Paraméterek:**

- **owner** (*String*): A kéz tulajdonosa (pl. *player*, *split*).

- **Példa:**

```
gameUI.disableHand("player");
```

void enableHand(String owner)

Engedélyezi a kéz kártyáit a felhasználói felületen.

- **Paraméterek:**

- **owner** (**String**): A kéz tulajdonosa (pl. **player**, **split**).

- **Példa:**

```
gameUI.enableHand("player");
```

void revealDealerCard()

Felfedi az osztó rejtett kártyáját.

- **Példa:**

```
gameUI.revealDealerCard();
```

void gameOver(String text)

Megjeleníti a játék végét jelző üzenetet.

- **Paraméterek:**

- **text** (**String**): A megjelenítendő üzenet.

- **Példa:**

```
gameUI.gameOver("Game Over: Player Wins!");
```

void updateBets(int wealthAmount, int betAmount)

Frissíti a fogadások és a játékos vagyonának értékét a felhasználói felületen.

- **Paraméterek:**

- **wealthAmount** (**int**): A játékos aktuális vagyona.
- **betAmount** (**int**): A játékos aktuális tétje.

- **Példa:**

```
gameUI.updateBets(1000, 50);
```

Példa használat

```
public class Main {  
    public static void main(String[] args) {  
        GameUI gameUI = new GameUI(new MyActionListener());  
        gameUI.drawCard(new Card(Rank.ACE, Suit.HEARTS,  
"/path/to/image.png"), "player");  
        gameUI.setEnableButton("deal");  
        gameUI.updateHandValue("player", "21");  
        gameUI.clear();  
        gameUI.clearPlayer();  
        gameUI.disableHand("player");  
        gameUI.enableHand("player");  
        gameUI.revealDealerCard();  
        gameUI.gameOver("Game Over: Player Wins!");  
        gameUI.updateBets(1000, 50);  
    }  
}
```

Main osztály

Áttekintés

A **Main** osztály a Blackjack játék indításáért felelős. Ez az osztály tartalmazza a **main** metódust, amely a program belépési pontja. A **main** metódus létrehoz egy új **Dealer** objektumot, és elindítja a játékot.

Osztály

Main

A Blackjack játék indításáért felelős osztály.

Konstruktor

Nincs konstruktor

A **Main** osztály nem tartalmaz konstruktort, mivel a **main** metódus a program belépési pontja.

Metódusok

```
public static void main(String[] args)
```


A program belépési pontja. Ez a metódus létrehoz egy új **Dealer** objektumot, hogy elindítsa a Blackjack játékot.

- **Paraméterek:**

- **args** (**String**[]): A parancssori argumentumok tömbje.

- **Példa:**

```
public static void main(String[] args) {  
    Dealer game = new Dealer();  
}
```

Példa használat

```
public class Main {  
    public static void main(String[] args) {  
        Dealer game = new Dealer();  
    }  
}
```

Felhasználói Kézikönyv

Bevezetés

Ez a kézikönyv a Blackjack játék használatát mutatja be. A játék célja, hogy a kártyák összértéke közelebb legyen a 21-hez, mint az osztóé, anélkül, hogy túllépné a 21-et.

Telepítés

1. Győződjünk meg róla, hogy a Java Development Kit (JDK) telepítve van a számítógépén.
2. Töltsük le a Blackjack játék forráskódját.
3. Nyissuk meg a parancssort vagy a terminált, és navigáljunk a letöltött forráskód könyvtárába.
4. Fordítsuk le a forráskódot a következő parancs segítségével:

```
javac -d bin src/main/java/blackjack/*.java
```

5. Futtassa a játékot a következő parancs segítségével:

```
java -cp bin blackjack.Main
```

Játék Kezdése

1. A játék indításakor egy új ablak jelenik meg, amely a Blackjack játék felhasználói felületét tartalmazza.
2. A játékos kezdeti egyenlege 2000 zseton.
3. A játékos beállíthatja a tét összegét a csúszka segítségével.
4. Nyomja meg a "Deal" gombot a kártyák kiosztásához.

Játék Menete

1. **Kártyák Kiosztása:** A játékos és az osztó két-két kártyát kap. Az osztó egyik kártyája lefelé fordítva marad.
2. **Játékos Döntései:**
 - **Hit:** A játékos további kártyát kér.
 - **Stay:** A játékos megáll, és nem kér több kártyát.
 - **Double:** A játékos megduplázza a tétet, és egy utolsó kártyát kér.
 - **Split:** Ha a játékos két azonos értékű kártyát kap, két külön kézre oszthatja őket.
3. **Osztó Játéka:** Miután a játékos befejezte a körét, az osztó felfedi a lefelé fordított kártyáját, és további kártyákat húz, amíg el nem éri a 17-et vagy annál magasabb értéket.
4. **Eredmény:** A játék véget ér, és az eredmény megjelenik a képernyőn. A játékos nyer, ha a kártyák összértéke közelebb van a 21-hez, mint az osztóé, anélkül, hogy túllépné a 21-et.

Gombok és Funkciók

- **Deal:** Kártyák kiosztása a játékosnak és az osztónak.
- **Hit:** További kártya kérése.
- **Stay:** Megállás, nem kér több kártyát.
- **Double:** Tét megduplázása és egy utolsó kártya kérése.
- **Split:** Két azonos értékű kártya két külön kézre osztása.

Játék Vége

A játék véget ér, ha a játékos vagy az osztó túllépi a 21-et, vagy ha mindkét fél megáll. Az eredmény megjelenik a képernyőn, és a játékos nyereménye vagy vesztesége frissül.

Kilépés

A játékból való kilépéshez zárja be az ablakot, vagy nyomja meg az "Esc" billentyűt.