Aufgabe 1

```
public class Intervall {
  // attributes
  public double a;
  public double b;
  private boolean empty = false;

  // constructors
  public Intervall (double a, double b) {
    if( a < b ) {
      this.a = a;
      this.b = b;
    } else {
      this.empty = true;
    }
  }

  // methods
  public boolean enthaelt(double x) {
    return !this.empty && (this.a <= x && x <= this.b);
  }
  public boolean schneidet(Intervall cd) {
    return !this.empty && ( (cd.a <= this.a && cd.b >= this.a) || (cd.a >= this.a
&& cd.a <= this.b) );
  }
  public boolean beinhaltet(Intervall cd) {
    return !this.empty && (cd.a >= this.a && cd.b <= this.b);
  }
  public double laenge() {
    if( !this.empty ) {
      return this.b-this.a;
    } else {
      return 0;
    }
  }
  public String toString() {
    if( !this.empty ){
      return String.format("Intervall [%s, %s]", this.a, this.b);
    } else {
      return "Intervall EMPTY";
    }
  }
}

public class TestIntervall {
  public static void main(String[] args) {
    Intervall i1 = new Intervall(1,5);
```

```
    Intervall i2 = new Intervall(-3,3);
    Intervall i3 = new Intervall(-3,-1);
    Intervall i4 = new Intervall(5,10);
    Intervall i5 = new Intervall(8,10);
    Intervall iempty = new Intervall(6,5);

    // Test enthaelt
    System.out.println(String.format("2.0 in %s?",i1));
    System.out.println(i1.enthaelt(2));
    System.out.println(String.format("1.0 in %s?",i1));
    System.out.println(i1.enthaelt(1));
    System.out.println(String.format("8.0 in %s?",i1));
    System.out.println(i1.enthaelt(8));
    System.out.println(String.format("2.0 in %s?",iempty));
    System.out.println(iempty.enthaelt(2));
    System.out.println();

    // Test schneidet
    System.out.println(String.format("%s schneidet %s?",i1,i4));
    System.out.println(i1.schneidet(i4));
    System.out.println(String.format("%s schneidet %s?",i4,i1));
    System.out.println(i4.schneidet(i1));
    System.out.println(String.format("%s schneidet %s?",i1,i5));
    System.out.println(i1.schneidet(i5));
    System.out.println(String.format("%s schneidet %s?",iempty,i1));
    System.out.println(iempty.schneidet(i1));
    System.out.println(String.format("%s schneidet %s?",i1,iempty));
    System.out.println(i1.schneidet(iempty));
    System.out.println();

    // Test beinhaltet
    System.out.println(String.format("%s beinhaltet %s?",i2,i3));
    System.out.println(i2.beinhaltet(i3));
    System.out.println(String.format("%s beinhaltet %s?",i3,i2));
    System.out.println(i3.beinhaltet(i2));
    System.out.println(String.format("%s beinhaltet %s?",i2,i1));
    System.out.println(i2.beinhaltet(i1));
    System.out.println(String.format("%s beinhaltet %s?",i2,iempty));
    System.out.println(i2.beinhaltet(iempty));
    System.out.println();

    // Test laenge
    System.out.println(i1);
    System.out.println(String.format("laenge: %s",i1.laenge()));
    System.out.println(i3);
    System.out.println(String.format("laenge: %s",i3.laenge()));
    System.out.println(iempty);
    System.out.println(String.format("laenge: %s",iempty.laenge()));
  }
}
```

Aufgabe 2

```java
import java.util.Random;

public class Gambler {

  // attributes
  private String name;
  private int id;
  private int startMoney;
  int currentMoney;
  int numBets;
  String gameProcess;

  // variables
  private static int nextId = 1;

  // constructors
  public Gambler(String name, int startMoney) {
    this.name = name;
    this.id = nextId;
    nextId ++;
    this.startMoney = startMoney;
    this.currentMoney = this.startMoney;
    this.numBets = 0;
    this.gameProcess = "";
  }
  private static String nextName() {
    return String.format("Gambler%s", nextId);
  }
  public Gambler(int startMoney) {
    this(nextName(), startMoney);
  }
  public Gambler() {
    this(25);
  }

  // standard methods
  public String toString() {
    return String.format("<Gambler> name: %s, start: %s$, current: %s$,
         plays: %s", this.name, this.startMoney, this.currentMoney, this.numBets);
  }

  // methods
  public void bet() {
    this.numBets ++;
```

```
    Random rand = new Random();
    if( rand.nextInt(2) == 0 ) {
      this.currentMoney ++;
      this.gameProcess += "+";
    } else {
      this.currentMoney --;
      this.gameProcess += "-";
    }
  }
  public String play() {
    String process = "";
    while( this.currentMoney > 0 ) {
      this.bet();
    }
    return this.gameProcess;
  }
}

public class Casino {
  // attributes
  private int size;
  private String name;
  Gambler[] gamblers;
  String gameProcesses;

  // constructor
  public Casino(int n, String name, int startMoney) {
    this.size = n;
    this.name = name;
    this.gamblers = new Gambler[n];
    for( int i=0 ; i<n; i++ ) {
      this.gamblers[i] = new Gambler(startMoney);
    }
    gameProcesses = "";
  }
  public Casino() {
    this(10, "Grand Royale Casino", 10);
  }

  // defaults
  public String toString() {
    String out = "<Casino>: " + this.name + "\n";
    for( int i=0 ; i<this.size ; i++ ) {
      out = out + this.gamblers[i] + "\n";
    }
    return out;
  }

  // methods
```

```
  public String getMoney() {
    for( int i=0 ; i<this.size ; i++ ) {
      String process = this.gamblers[i].play();
      this.gameProcesses += String.format("%s\n%s\nLost all his money\n\n",
                                                    this.gamblers[i], process);
    }
    return this.gameProcesses;
  }
}

public class Test {
  public static void main(String[] args) {
    // Gambler Tests
    Gambler g1 = new Gambler("Tobias", 10);
    System.out.println(g1);
    System.out.println(g1.play());
    System.out.println(g1);
    System.out.println();
    Gambler g2 = new Gambler();
    System.out.println(g2);
    System.out.println(g2.play());
    System.out.println(g2);
    System.out.println();

    // Casino Tests
    Casino casino = new Casino();
    System.out.println(casino);
    System.out.println(casino.getMoney());
    System.out.println(casino);
  }
}
```

Aufgabe 3 & 4

```
import java.util.Random;

public class Position {
  // attributes
  int holeNeighbours = 0;
  boolean hole = false;
  boolean open = false;

  // constructor
  private static final Random RAND = new Random();
  public Position(double p) {
    if( RAND.nextDouble() <= p ) {
      this.hole = true;
```

```
    }
  }
}

public class PlayField {
  // attributes
  Position[][] field;
  final int n;
  final int m;

  // constructor
  public PlayField(int n, int m, double p) {
    this.n = n;
    this.m = m;
    // create clear field
    field = new Position[n+2][m+2];
    // initialize field
    for( int i=0 ; i<=n+1 ; i++ ) {
      for( int j=0 ; j<=m+1 ; j++ ) {
        // makes sure that border has no holes
        if( !inField(i,j) ) {
          field[i][j] = new Position(0);
        } else {
          field[i][j] = new Position(p);
        }
      }
    }
    // generate solution
    for( int i=1 ; i<=n ; i++ ) {
      for( int j=1 ; j<=m ; j++ ) {
        calculateNeighbours(i,j);
      }
    }
  }
  public PlayField() {
    this(15, 20, 0.1);
  }

  // default methods
  public String toString() {
    String out = "";
    for( int i=1 ; i<=n ; i++ ) {
      for( int j=1 ; j<=m ; j++ ) {
        if( field[i][j].open ) {
          if( field[i][j].hole ) {
            out += "\u274D ";
          } else if( field[i][j].holeNeighbours == 0 ) {
            out += ". ";
          } else {
```

```
          out += field[i][j].holeNeighbours + " ";
        }
      } else {
        out += "\u25FB ";
      }
    }
    out += "\n";
  }
  return out;
}

// private methods
private void calculateNeighbours(int i, int j){
  for( int k=i-1 ; k<=i+1 ; k++ ) {
    for( int l=j-1 ; l<=j+1 ; l++ ) {
      if( field[k][l].hole ) {
        field[i][j].holeNeighbours ++;
      }
    }
  }
}
private boolean inField(int i, int j) {
  return i>=1 && i<=n && j>=1 && j<=m;
}

// public methods
public void expose() {
  for( int i=1 ; i<=n ; i++ ) {
    for( int j=1 ; j<=m ; j++ ) {
      field[i][j].open = true;
    }
  }
}
public boolean updateField(int i, int j) {
  boolean playing;
  if( field[i][j].hole ) {
    expose();
    return false;
  } else if( field[i][j].holeNeighbours > 0 ) {
    field[i][j].open = true;
    return true;
  } else {
    field[i][j].open = true;
    for( int k=i-1 ; k<=i+1 ; k++ ) {
      for( int l=j-1 ; l<=j+1 ; l++ ) {
        if( inField(k,l) && !(k==i && l==j) && !field[k][l].open ) {
          updateField(k,l);
        }
      }
    }
```

```
      return true;
    }
  }
  public boolean solved() {
    boolean solution = true;
    int i = 1;
    while( solution && i<=n ) {
      int j = 1;
      while( solution && j<=m ) {
        solution = solution && (field[i][j].open ||
                                (!field[i][j].open && field[i][j].hole) );
        i++;
      }
      j++;
    }
    return solution;
  }
}

public class TestPlayField {
  public static void main(String[] args) {
    PlayField field = new PlayField();
    System.out.println(field);
    field.expose();
    System.out.println(field);
  }
}

public class Player {
  // attributes
  final String name;
  int score = 0;
  int rounds = 0;
  boolean playing = true;
  boolean won = false;
  PlayField playField = new PlayField();

  // constructors
  private static int nextPlayer = 1;
  public Player() {
    this.name = "Player"+ nextPlayer;
    nextPlayer++;
  }

  // default methods
  public String toString() {
    return String.format("%s  score: %s", playField, score);
  }
```

```
  // private methods
  private void updateScore() {
    score = 0;
    for( int i=1 ; i<=playField.n ; i++ ) {
      for( int j=1 ; j<=playField.m ; j++ ) {
        if( playField.field[i][j].open ) {
          score += playField.field[i][j].holeNeighbours;
        }
      }
    }
  }

  // public methods
  public void pick(int i, int j) {
    rounds++;
    playing = playField.updateField(i, j);
    if( playing ){
      updateScore();
      if( playField.solved() ) {
        playing = false;
        won = true;
      }
    }
  }
  private static final Random RAND = new Random();
  public String randomPick() {
    int i = 0;
    int j = 0;
    while( !playField.field[i][j].open ) {
      i = RAND.nextInt(playField.n)+1;
      j = RAND.nextInt(playField.m)+1;
      pick(i, j);
    }
    return String.format("  round: %s, picked: (%s,%s)\n", rounds, i, j);
  }
  public int play() {
    while( playing ) {
      String pick = randomPick();
      System.out.println(pick + this);
    }
    String out = won? "\n  WON\n\n" : "\n  LOST\n\n";
    System.out.println(out);
    return score;
  }
}

public class HolePlay {
  // attributes
  Player[] player;
```

```
  int size;

  // constructor
  public HolePlay(int n) {
    this.player = new Player[n];
    this.size = n;
    for( int i=0 ; i<size ; i++ ) {
      player[i] = new Player();
    }
  }

  // default methods
  public String toString() {
    String out = String.format("Game with %s Players:\n", size);
    for( int i=0 ; i<size ; i++ ) {
      out += String.format("%s score: %s\n", player[i].name, player[i].score);
    }
    return out;
  }

  // methods
  public void simulatePlay() {
    for( int i=0 ; i<size ; i++ ) {
      System.out.println(player[i].name + "\n");
      player[i].play();
    }
    Player winner = maxPlayerByScore();
    System.out.println(winner.name + " won with a score of " +
                                                    winner.score + ".\n\n");
  }
  public Player maxPlayerByScore() {
    Player max = player[0];
    for( int i=1 ; i<size ; i++ ) {
      max = max.score>player[i].score ? max : player[i];
    }
    return max;
  }

  // main
  public static void main(String[] args) {
    HolePlay thisPlay = new HolePlay(5);
    thisPlay.simulatePlay();
  }
}
```