Universität der Bundeswehr München
Fakultät für Informatik – Institut für Softwaretechnologie
Prof. Dr.-Ing Mark Minas

Klausur
Objektorientierte Programmierung

16.09.2020

Folgende Anweisungen sind zu beachten:

- Diese Klausur umfasst insgesamt **8** Seiten. Prüfen Sie, ob Ihre Unterlagen vollständig und alle Seiten einwandfrei zu lesen sind.
- Schreiben Sie in alle dafür vorgesehenen, umrahmten Felder Ihren Namen und Ihre Matrikelnummer. Aufgabenblätter ohne diese Angaben werden in der Korrektur nicht berücksichtigt.
- Verwenden Sie zum Lösen der Aufgaben entweder einen Füllfederhalter oder einen Kugelschreiber in blauer oder schwarzer Farbe. Verwenden Sie keine anderen Farben. Verwenden Sie insbesondere weder Bleistift noch Rotstift (auch nicht für Streichungen). Die Verwendung von Mitteln, welche die Dokumentenechtheit verletzen, z.B. Tipp-Ex oder Tintenkiller, ist ebenfalls untersagt. Bei Nichtbeachtung werden die entsprechenden Lösungen nicht gewertet.
- Pro Aufgabe kann nur eine Lösung gewertet werden. Ungültige Lösungen sind erkennbar **durchzustreichen**.

Kenntnis genommen

_____

(Name in **Blockschrift**, Matrikelnummer, Unterschrift)

Viel Erfolg!

	Testate	Aufg. 1	Aufg. 2	Aufg. 3	Aufg. 4	Summe
max. Punkte:	18	8	8	15	14	63
Erzielte Punkte:						

In dieser Klausur geht es um das effiziente Speichern von Punkten in der Ebene. Im Folgenden gehen wir davon aus, dass alle Punkte (x, y) ganzzahlige Koordinaten $x, y \in \mathbb{Z}$ besitzen und $0 \leq x, y < 2^n$ gilt (**beachte** das $<$-Zeichen!), wobei n eine beliebige, aber feste **positive** ganze Zahl ist. Damit liegen alle Punkte in einem Quadrat der Kantenlänge 2^n.

Sei nun eine Menge P solcher Punkte gegeben. Um P effizient zu speichern, zerlegt man das sie enthaltende Quadrat sukzessive in kleinere Quadrate, bis jedes dieser Quadrate genau einen oder keinen Punkt enthält. Dazu geht man folgendermaßen vor: Enthält ein Quadrat mehr als einen Punkt, zerlegt man es in vier gleich große Quadrate („Quadranten“). Anschließend verfährt man mit jedem der Quadranten in derselben Weise, d.h. man zerlegt einen Quadranten erneut, wenn er mehr als einen Punkt enthält usf. Am Ende dieses Vorgehens ist das ursprüngliche Quadrat der Kantenlänge 2^n mit Quadraten parkettiert, die alle maximal einen Punkt enthalten.

Das folgende Beispiel (Abb. 1) zeigt eine Parkettierung des Quadrats mit Kantenlänge 8 für eine Punktmenge $P = \{(1, 7), (4, 1), (5, 2)\}$.

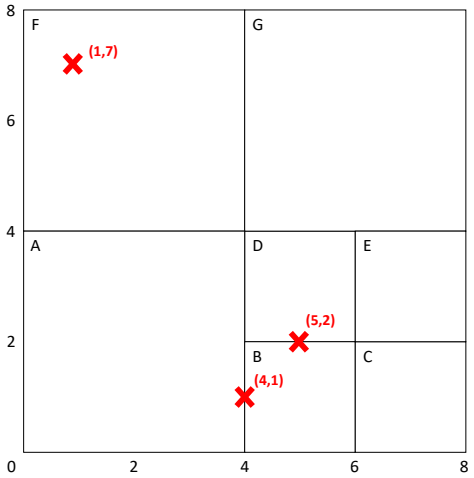

Abbildung 1: Parkettierungsbeispiel

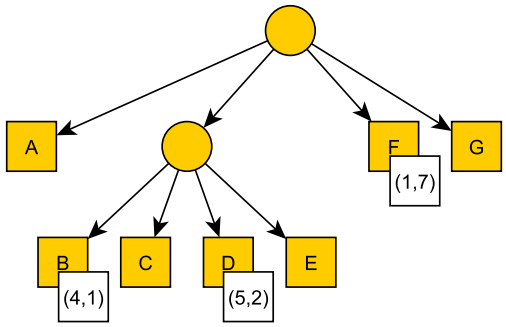

Abbildung 2: Quadtree für Abb. 1.

Die Quadrate B, D, F enthalten jeweils genau einen Punkt, wohingegen A, C, E, G leer sind. Beachte, dass Punkte auf der linken und unteren Begrenzungslinie eines Quadrats liegen dürfen, aber nicht auf der rechten oder oberen. Damit liegt der Punkt $(4, 1)$ im Quadrat B und $(5, 2)$ in D. Beachte außerdem, dass jedes der Quadrate eine Kantenlänge 2^m mit $m \leq n$ hat.

Eine solche Punktmenge P und eine dazugehörige Parkettierung kann effizient mit einem sogenannten *Quadtree* repräsentiert werden. Ein Quadtree ist ein Baum, dessen Knoten jeweils ein Quadrat repräsentieren. Wie bei Bäumen üblich, unterscheidet man *innere Knoten* und *Blätter*. Ein innerer Knoten eines Quadtrees hat genau vier Sohnknoten; er repräsentiert damit ein Quadrat, das in seine vier Quadranten zerlegt wurde. Blätter verweisen entweder auf genau *einen Punkt* oder sind *leer*.

Der in Abb. 2 dargestellte Quadtree repräsentiert die in Abb. 1 gezeigte Parkettierung zur Punktmenge P. Die Blattknoten sind mit $A, B, C, \ldots$ benannt und repräsentieren die entsprechend benannten Quadrate. Die Sohnknoten eines inneren Knotens sind immer in derselben Reihenfolge angeordnet: zuerst die beiden unteren Quadranten von links nach rechts, dann die beiden oberen.

Name:	Matrikelnummer:

Aufgabe 1 (Darstellung von Punkten): **8 Punkte**

Schreiben Sie die Klasse `Point` in Java zur Realisierung von Punkten (x, y) mit ganzzahligen Koordinaten $x, y \in \mathbb{Z}$. Der Punkt $(4, 1)$ soll beispielsweise mit dem Ausdruck `new Point(4,1)` erstellt werden. Alle Attribute sollen privat deklariert sein. Schreiben Sie alle notwendigen Methoden, damit auf die Attribute lesend zugegriffen werden kann und `Point`-Objekte in naheliegender Weise auf Gleichheit geprüft werden können. Ferner soll `System.out.println(p)` eine sinnvolle Ausgabe produzieren, wenn `p` vom Typ `Point` ist, und es möglich sein, `Point`-Objekte in Hashtabellen zu speichern.

```
public class Point {
```

Name:	Matrikelnummer:

Aufgabe 2 (Quadrate): 6 + 2 = **8 Punkte**

a) Vervollständigen Sie die Klasse `Square` zur Realisierung von Quadraten, wie sie für die zuvor beschriebene Parkettierung benötigt werden. Die Attribute `left` und `right` stehen für die x-Koordinaten des linken und rechten Randes des Quadrats, `bottom` und `top` für die y-Koordinaten des unteren und oberen Randes. Das Objekt für das Quadrat E in Abb. 1 werde z.B. mit `new Square(6,8,2,4)` erzeugt. Implementieren Sie insbesondere die Methoden `bottomLeft()` etc., die die vier Quadranten zurückgeben. So soll mit `new Square(0,8,0,8).topLeft()` ein Objekt für den linken oberen Quadranten des Gesamtquadrats, d.h. F zurückgegeben werden.

```
public class Square {
  private final int left, right, bottom, top;


  public Square(int left, int right, int bottom, int top) {









  }

  public Square bottomLeft() {

  public Square bottomRight() {

  public Square topLeft() {

  public Square topRight() {
```

b) Ergänzen Sie die Klasse *Square* um die Methode *contains*, welche prüft, ob ein gegebener Punkt in diesem Quadrat liegt oder nicht.

Name:	Matrikelnummer:

Aufgabe 3 (Knoten von Quadtrees): $5 + 2 + 3 + 5 =$ **15 Punkte**

Die abstrakte Klasse `Node` repräsentiert Knoten von Quadtrees und damit Quadrate.

```
public abstract class Node {
   public final Square square;
   public Node(Square square) { this.square = square; }
   public abstract Node add(Point point);
}
```

Die Ausprägung als Blätter und innere Knoten erfolgt in Form von konkreten Unterklassen `EmptyNode`, `PointNode` und `InnerNode`. Die ersten beiden Klassen repräsentieren Blätter, deren Quadrate keinen bzw. genau einen Punkt enthalten. Die drei Unterklassen müssen die Methode `add` implementieren. In dieser Aufgabe müssen Sie deren Implementierung aber noch nicht angeben; das ist Thema der Aufgabe 4.

a) Zeichnen Sie das zugehörige UML-Klassendiagramm auf Seite 6, wobei Sie die Methoden der einzelnen Klassen weglassen können.

b) Vervollständigen Sie die nachfolgende Klasse `EmptyNode` (bis auf die entsprechend kommentierten Methoden). Sie repräsentiert Blätter, deren Quadrate keinen Punkt enthalten.

```
public class EmptyNode

  @Override
  public Node add(Point point) {...} // siehe Aufgabe 4
  @Override
  public String toString() {...} // muss nicht implementiert werden
```

c) Vervollständigen Sie die nachfolgende Klasse `PointNode` (bis auf die entsprechend kommentierten Methoden). Sie repräsentiert Blätter, deren Quadrate jeweils genau einen Punkt enthalten. Dieser Punkt muss natürlich im Objekt dieser Klasse geeignet gespeichert werden.

```
public class PointNode

  @Override
  public Node add(Point point) {...} // siehe Aufgabe 4
  @Override
  public String toString() {...} // muss nicht implementiert werden
```

d) Vervollständigen Sie nun die Klasse `InnerNode` (bis auf die entsprechend kommentierten Methoden); sie repräsentiert innere Knoten. Die vier Sohnknoten sind im Attribut `children` zu speichern. Sorgen Sie dafür, dass die vier Sohnknoten unmittelbar nach Erstellung einer `InnerNode`-Instanz jeweils Blätter ohne einen Punkt, d.h. `EmptyNode`-Instanzen sind. Die Söhne im Array sollen in der zuvor beschriebenen Reihenfolge abgelegt werden, d.h. zuerst die für die beiden unteren Quadranten, dann die für die beiden oberen, und jeweils von links nach rechts.

Hinweis: Sie können die Methoden `topLeft()` etc. und die Methode `contains()` der Klasse `Square` verwenden.

```
public class InnerNode

  private final Node[] children;
```

```
@Override
public Node add(Point point) {...} // siehe Aufgabe 4
@Override
public String toString() {...} // muss nicht implementiert werden
```

Name:	Matrikelnummer:

UML-Klassendiagramm zu Teilaufgabe a):

Name:	Matrikelnummer:

Aufgabe 4 (Punkte hinzufügen): 3 + 6 + 5 = **14 Punkte**

In dieser Aufgabe geht es um die Implementierung der Methode `add` in den drei Unterklassen von `Node`. Damit wird ein neuer Punkt in den Quadtree eingefügt, wodurch gegebenenfalls Quadrate in Quadranten zerlegt werden müssen.

Beispiel: Im Folgenden sind zwei Quadtrees dargestellt. Der in Abb. 3 entsteht aus dem in Abb. 2 durch Hinzufügen des Punkts $(5, 5)$. Dadurch muss das zuvor leere Blatt G in eines geändert werden, das den neuen Punkt $(5, 5)$ enthält.

Der zweite Baum (Abb. 4) entsteht durch anschließendes Hinzufügen des Punkts $(1, 6)$. Dazu muss das Quadrat F zerlegt werden, d.h. aus dem Blatt F wird ein neuer innerer Knoten. Aber da sowohl $(1, 7)$ als auch $(1, 6)$ in dessen linkem oberen Quadranten liegen, muss man diesen erneut zerlegen. Dadurch entstehen insgesamt zwei neue innere Knoten.

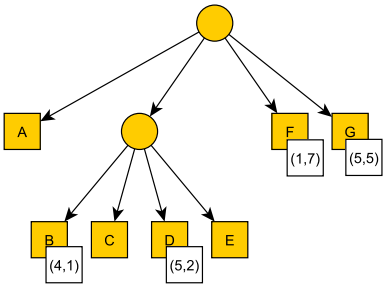

Abbildung 3: Ergebnis nach Einfügen von $(5, 5)$ in Abb. 2.

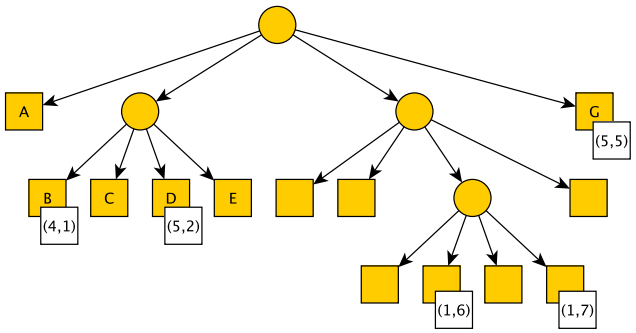

Abbildung 4: Ergebnis nach Einfügen von $(1, 6)$ in Abb. 3.

Sei `node` ein Knoten. Der Aufruf `node.add(point)` fügt dann den Punkt `point` in den Baum ein, an dessen Wurzel `node` steht. Wenn der Knoten `node` dadurch geändert werden muss (weil z.B. aus einem `EmptyNode` ein `PointNode` wird), muss `add` den neu erstellten Knoten zurückgeben, andernfalls `node`. Liegt `point` gar nicht im Quadrat von `node`, soll eine `IllegalArgumentException` geworfen werden.

Hinweis: Beachten Sie bei der Erzeugung von Quadraten die Definition der Grenzen dieser Quadrate.

a) Schreiben Sie die `add`-Methode der Klasse `EmptyNode`

b) Schreiben Sie die `add`-Methode der Klasse `PointNode`. Beachten Sie, dass der Baum nicht geändert werden muss, wenn der „hinzugefügte“ Punkt tatsächlich schon enthalten ist.

c) Schreiben Sie die `add`-Methode der Klasse `InnerNode`.