- Implementierung von parkeZug(z:Zug) (4 VP)

```
public void parkeZug(Zug z) {
  BinaryTree b = this.gleise;
  while (b.getLeft() != null) {
    Weiche w = (Weiche)b.getContent();
    if (w.isGerade() == true) {
      b = b.getRight();
    } else {
      b = b.getLeft();
    }
  }
  b.setContent(z);
}
```

- Laufzeitverhalten (2 VP)
  Bei geschickter Planung der Gleisanlagen (vollständiger balancierter Binärbaum) beträgt die Laufzeit im schlimmsten Fall (und auch im besten Fall) O(log n), da in einem Baum der Tiefe log(n) maximal n Blätter untergebracht werden können. Daher muss der Zug dann im schlimmsten Fall an O(log n) Weichen vorbeifahren. Ist die Gleisstruktur allerdings degeneriert, so dass alle Weichen hintereinander liegen, dann beträgt die Laufzeit im schlimmsten Fall O(n).

B3.3 • Implementierung von rangiereZug(z:Zug) (3 VP)

```
public void rangiereZug(Zug z) {
  while (!z.isEmpty()) {
    Waggon w = z.pop();
    stelleWeichenZumZug(this.gleise, w.getZiel());
    lasseWaggonRollen(w);
    }
}
```

Aufgabe II B3:

B3.1 • Beschreibung des ADT Stapel (3 VP)
Die Methode push nimmt als Parameter ein Objekt der Klasse Container entgegen und legt es als oberstes Element auf dem Stapel ab. Die Methode pop entfernt das oberste Objekt vom Stapel und liefert es als Ergebnis zurück.
Ein Stapel (Keller) ist eine lineare Datenstruktur. Elemente können ihr nur auf einer Seite hinzugefügt werden, und nur von dieser Seite können sie auch wieder entnommen werden. Dieses Verfahren entspricht dem LIFO-Prinzip (Last In – First Out).
Schiffscontainer können auf einem Stapel nur oben abgelegt werden (push) bzw. nur von oben aus dem Stapel entfernt werden (pop). Der ADT Stapel ist somit eine geeignetes Modell für einen Containerstapel.

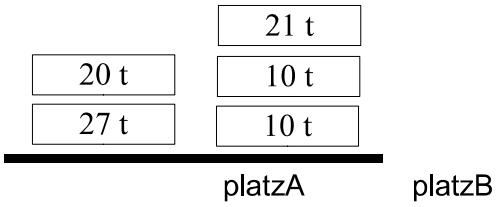
- Implementierung von getGesamtgewicht() (3 VP)

```
public double getGesamtgewicht() {
  LinkedList liste = first;
  double gewicht = 0.0;
  while (liste != null) {
    Container c = (Container) liste.getContent();
    gewicht = gewicht + c.getGewicht();
    liste = liste.getNext();
  }
  return gewicht;
}
```

B3.2 • Implementierung von aufladen(neu:Container) (4 VP)

```
public void aufladen(Container neu) {
  if (platzA.getGesamtgewicht() < platzB.getGesamtgewicht())
    platzA.push(neu);
  else
    platzB.push(neu);
}
```

B3.3 • Skizze der Ladesituation (2 VP)



- Vergleich der Algorithmen (2 VP)
  Der Algorithmus der einfachen Methode aufladen(..) führt zu diesem Ergebnis:

Zeitschritt	Gesamt-gewicht A	Gesamt-gewicht B	Gewicht neuer Container C	Entscheidung
0	10	10	21	Der neue Container kommt immer auf den Stapel mit dem geringeren Gesamtgewicht.
1	10	31	27	
2	37	31	20	
3	37	51		

Die folgenden Werte erhält man mit dem Algorithmus der experimentellen Methode aufladen2(..):

Zeitschritt	Gesamt-gewicht A	Gesamt-gewicht B	Gewicht neuer Container c	A < B	A + C > B + 20	B + C > A +20
0	10	10	21	nein	ja	–
1	21	20	27	nein	ja	–
2	27	41	20	ja	–	nein
3	47	41				

Im vorgegeben Beispiel führt Algorithmus 2 zu einer gleichmäßigeren Verteilung auf den beiden Ladeplätzen.

- Berurteilung der experimentellen Lademethode (2 VP)

  Zu manchen Zeitpunkten bringt Algorithmus 2 auch deutliche Verschlechterungen: Bei den für die Aufgabe gewählten Gewichten entsteht in Zeitschritt 2 in der einfachen Lademethode nur eine Gewichtsdifferenz von 6 t, bei der neuen Lademethode beträgt sie 14 t.

  Die Bewertung ergibt, dass eine Übernahme der experimentellen Lademethode in den Produktivbetrieb nicht zu empfehlen ist.

  Hinweis: zu dieser Empfehlung können auch selbst gewählte Zahlenbeispiele führen. So sind deutlichere Nachteile möglich, wie die Ladung eines 21 t - Containers auf zwei Stapel mit je einem 28 t - Container zeigt. Hier steigt bei Algorithmus 2 die Gewichtsdifferenz auf 35 t, während sie sonst bei den 21 t des letzten Containers liegt. Zur Begründung der abschließenden Empfehlung genügt die Angabe eines korrekten Zahlenbeispiels.

B3.4 • Neuverkettung der Objekte (4 VP)

Bei leerer Liste verweist der Listenkopf first nach dem Einfügen auf das neue Listenelement e. Der Nachfolger von e bleibt leer (null).

Ist die Liste nicht leer, wird nach dem ersten Listenelement f gesucht, das einen Container für einen später zu bedienenden Hafen enthält. Der Vorgänger v von f, der auch der Listenkopf first sein kann, erhält dann als neuen Nachfolger das einzufügende Element e. Der bisherige Nachfolger von v wird zum Nachfolger von e.

Visualisierung: