Vorgehensweise und Entscheidungsfindung

Nach dem Import der Klassen „Pile1Test“, „Pile2Test“ und „IPile“ habe ich aus dem Interface „IPile“ die Klassen „Pile1“ und „Pile2“ samt Methodenrümpfen generiert.
Als Attribute haben die beiden Klassen jeweils ein int-Array „a“ und ein int „n“ für die Länge des spezifischen Arrays.

In den Konstruktoren der Klassen „Pile1“ und „Pile2“ wird das Attribut „n“ mit dem Ergebnis von 2 potenziert mit dem Übergabeparameter des Konstruktors initialisiert. Anschließend initialisiere ich das Array „a“ mit der soeben festgelegten Länge „n“ (n ist final deklariert, da sich die Arraylänge nicht mehr ändern kann, daher sollte sich dieser Wert auch nicht mehr ändern, da er später genutzt wird). In der folgenden for-Schleife wird das Array mit Zahlen von 0,1,2,3, -> n-1 (Arraylänge) gefüllt.

Anschließend habe ich die Methoden nach den gegebenen Vorgaben ausformuliert.

Nach erfolgreichem Test mit den JUnit-Tests, habe ich die beiden Klassen „PileIO“, für Ein- und Ausgaben, und „PileRun“, zum starten und steuern des Programms, erstellt.
Über „PileIO“ werden in den beiden Methoden einmal die Potenz zur Basis zwei eingelesen und das gewünschte „j“ für Aufgabenteil vier. Weiterhin gibt es eine Methode, die einen char einliest, dieser wird in der Klasse „PileRun“ benötigt, um das Programm zu beenden oder die Endlosschleife fortzuführen (for(;;)).

class PileRun

In der Klasse „PileRun“ befindet sich die main-Methode. Es werden hier nacheinander Objekte der Klassen „Pile1“ und „Pile2“ erzeugt. Im Konstruktoraufruf wird eine gewünschte 2er Potenz an Objekte der Klasse „Pile1“ und „Pile2“ übergeben. Anschließend wird die Methode „inorder“ aus den jeweiligen Klassen „Pile1“ und „Pile2“ aufgerufen und zwar mit dem gewünschten Übergabewert, der direkt zuvor von der Tastatur eingelesen wird.
Auf der Konsole folgt umgehend die Ausgabe.

class Pile1 und Pile2

Die rekursive Methode „inorder“ prüft zunächst, ob der Übergabeparameter „j“ größer ist, als die Arraylänge-1. Ist dies der Fall, gibt die Methode eine Meldung auf der Konsole aus und beendet das Programm im Anschluss, um einer „ArrayOutOfBound“ Exception zu entgehen. Ist der Wert gültig, werden im Anschluss „leftChild“ und „rightChild“ von „j“ berechnet und in lokalen Variablen zwischengespeichert. Nun wird geprüft, ob „leftChild“ ungleich 0 ist, ist dies der Fall, wird erneut „inorder“ aufgerufen, aber mit dem eben errechneten und gespeicherten Wert für „leftChild“. Danach wird die Methode „visit“ mit dem vorherigen „j“ aufgerufen. Im Anschluss wird geprüft, ob es ein „rightChild“ gibt, wenn ja wird „inorder“ mit dem gespeicherten Wert aufgerufen.
Beginnt man bei Pile1 „oben“ mit inorder, also z.B. inorder(1) bei einem Pile1 von 2^3, wird der Baum komplett von 1->2^3-1 durchlaufen. Fängt man mit einem höheren j an, gibt es entsprechend weniger Durchläufe bei inorder.
Beginnt man bei Pile2 „oben“, also z.B. mit inorder(4), bei einem Pile2 von 2^3, wird der Baum komplett von 4->1 und von 4->7 durchlaufen, bei einem niedrigeren j, z.B. inorder(1), gibt es kein left oder righChild, es folgt nur ein Durchlauf und visit gibt 1 zurück (parent).

Sibling ist konsistent zu den leftChilds und rightChilds, nicht aber zu den parents. leftChild ist Nachbar von rightChild und umgekehrt, parent ist aber kein Nachbar von beiden.

Quellenangaben

Ideenaustausch über die Konzeption fand statt mit folgenden Kommilitonen:
Tristan Rudat
Marcel Ehrlitzer

Math Klasse
Sun Microsystems, Inc.; Class Math
URL: http://java.sun.com/j2se/1.4.2/docs/api/java/lang/Math.html
(abgerufen am 17.04.2010)