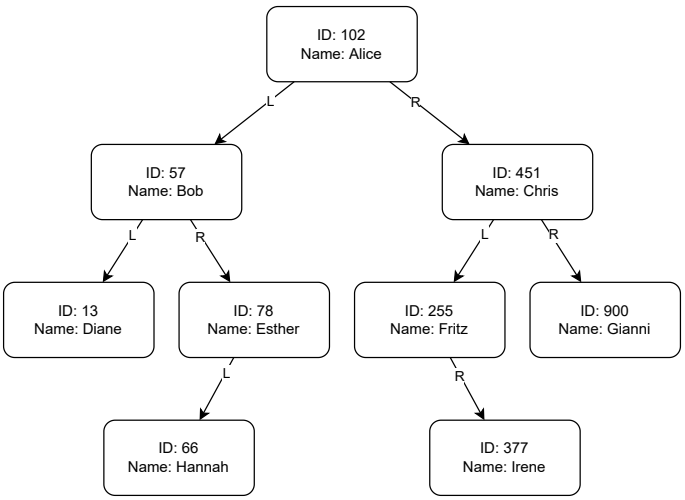
4 Datenstrukturen

Ein Binärbaum ist eine Datenstruktur, welche die Suche von Daten beschleunigt. Ein Binärbaum besteht aus Knoten, die Daten speichern können. Jeder Knoten kann einen linken und einen rechten Kindknoten besitzen. Der linke Kindknoten enthält einen Wert der kleiner oder gleich dem Wert des Wurzelknotens ist, und der rechte Kindknoten enthält einen Wert der größer ist.

Im folgenden sollen Sie eine Personendatenbank schreiben, die mit einem Binärbaum verwaltet wird. Jeder Person wird eine zufällige ID zwischen 1 und 1000 zugeordnet, welche im Binärbaum zur Suche verwendet werden soll. Der entstehende Binärbaum kann beispielsweise wie folgt aussehen:



Aufgaben

1. Erstellen Sie ein Projekt und initialisieren Sie es mit einer Git-Repository. Legen Sie zu jedem der folgenden Aufgaben einen Commit an sobald Sie diese gelöst haben.
2. Legen Sie die Klasse `Person` mit den Feldern *Name* und einem Integer *ID* an.

Lösung:

Person.java

```java
import java.util.Random;

public class Person {
    private final String name;
    private final int id;

    public static final int MAX_RAND_ID = 1000;

    public Person(String name, int id) {
```

```
        this.name = name;
        this.id = id;
    }

    public Person(String name) {
        this.name = name;
        this.id = new Random().nextInt(MAX_RAND_ID);
    }

    public String getName() {
        return name;
    }

    public int getId() {
        return id;
    }
}
```

3. Erstellen Sie einen Binärbaum, in dem Personen hinterlegt werden können. Nachdem eine Person dem Binärbaum hinzugefügt wurde, soll diese mittels der ID wieder zurückgegeben werden können. Legen Sie hierfür die Klasse `Binaerbaum` an, welche die Methoden *hinzufuegen(p: Person): void* und *finden(id: int): Person* besitzt.

   Hinweis: Binärbäume sind rekursiv! Die beiden Kindknoten einer Wurzel sind wiederum selber die Wurzeln eines Binärbaumes.

Lösung:

Binaerbaum.java

```
public class Binaerbaum {
    private Node wurzelknoten;

    public void hinzufuegen(Person p) {
        if(wurzelknoten == null) {
            wurzelknoten = new Node(p);
        } else {
            wurzelknoten.hinzufuegenRekursiv(p);
        }
    }

    /**
     *
     * @return The person with the ID, or null if there is no such
↪   person
     */
```

```
    public Person finden(int id) {
        if(wurzelknoten == null)
            return null;
        else {
            return wurzelknoten.findenRekursiv(id);
        }
    }

}
```

Node.java

```
public class Node {

    private Node links, rechts;
    private Person person;

    public Node(Person p) {
        this.person = p;
    }

    public void hinzufuegenRekursiv(Person einfuegePerson) {
        if(einfuegePerson.getId() <= person.getId()) {
            if(links != null) {
                links.hinzufuegenRekursiv(einfuegePerson);
            } else {
                links = new Node(einfuegePerson);
            }
        } else {
            if(rechts != null) {
                rechts.hinzufuegenRekursiv(einfuegePerson);
            } else {
                rechts = new Node(einfuegePerson);
            }
        }
    }

    public Person findenRekursiv(int id) {
        if(id == person.getId())
            return person;
        else if(id < person.getId()) {
            if(links != null)
                return links.findenRekursiv(id);
            else
                return null;
        } else {
```

```
            if(rechts != null)
                return rechts.findenRekursiv(id);
            else
                return null;
        }
    }
}
```

4. Ändern Sie Ihre Klasse für Binärbäume so ab, dass beliebige Typen gespeichert werden können, solange diese das Interface `Comparable` implementieren. Nutzen Sie dafür Javas generische Typisierung.

Tipp

Keine Angst Code zu löschen! Mit Git kann immer zu dem Stand eines Commits zurückgesprungen werden.

Lösung:

Person.java

```
import java.util.Random;

public class Person implements Comparable<Person> {
    private final String name;
    private final int id;

    public static final int MAX_RAND_ID = 1000;

    public Person(String name, int id) {
        this.name = name;
        this.id = id;
    }

    public Person(String name) {
        this.name = name;
        this.id = new Random().nextInt(MAX_RAND_ID);
    }

    public String getName() {
        return name;
    }

    public int getId() {
        return id;
    }
```

```
    @Override
    public int compareTo(Person anderePerson) {
        int andereID = anderePerson.getId();
        return id - andereID;
    }
}
```

Binaerbaum.java

```
public class Binaerbaum<T extends Comparable<T>> {
    private Node<T> wurzelknoten;

    public void hinzufuegen(T p) {
        if(wurzelknoten == null) {
            wurzelknoten = new Node<T>(p);
        } else {
            wurzelknoten.hinzufuegenRekursiv(p);
        }
    }

    public T finden(T findeObj) {
        if(wurzelknoten == null)
            return null;
        else {
            return wurzelknoten.findenRekursiv(findeObj);
        }
    }

}
```

Node.java

```
public class Node <E extends Comparable<E>> {

    private Node<E> links, rechts;
    private E value;

    public Node(E val) {
        this.value = val;
    }

    public void hinzufuegenRekursiv(E einfuegeVal) {
        if(value.compareTo(einfuegeVal) < 0) {
            if(links != null) {
                links.hinzufuegenRekursiv(einfuegeVal);
```

```
            } else {
                links = new Node<E>(einfuegeVal);
            }
        } else {
            if(rechts != null) {
                rechts.hinzufuegenRekursiv(einfuegeVal);
            } else {
                rechts = new Node<E>(einfuegeVal);
            }
        }
    }

    public E findenRekursiv(E findeObj) {
        if(value.compareTo(findeObj) == 0)
            return value;
        else if(value.compareTo(findeObj) < 0) {
            if(links != null)
                return links.findenRekursiv(findeObj);
            else
                return null;
        } else {
            if(rechts != null)
                return rechts.findenRekursiv(findeObj);
            else
                return null;
        }
    }
}
```