

Fortsetzung 2. Aufgabe

Korrekturrand

b) In der „BESUCHERAPP“ soll es zunächst drei Typen von Besuchern geben.

- „STANDARD“-Besucher zahlen für Online-Tickets den regulären Eintrittspreis und haben nur über den Haupteingang Zutritt zum Park.
- „PREMIUM“-Besucher bekommen 5 Prozent Rabatt auf den regulären Eintrittspreis, sowie an Werktagen auch Zutritt über den Expresseingang.
- „VIP“-Besuchern wird 10 Prozent Nachlass und an allen Tagen Zutritt über den Expresseingang gewährt.

In einer ersten Implementierung enthält die Klasse Besucher Methoden mit redundanten Auswahlstrukturen.

Besucher
- typ : String
+ Besucher(typ : String)
+ isExpressEingang(isWerktag : Boolean) : Boolean
+ calculatePreis(basisPreis : Double) : Double

+ isExpressEingang(isWerktag : Boolean) : Boolean

"STANDARD"	"PREMIUM"	typ
Rückgabe: false	Rückgabe: isWerktag	Rückgabe: true

+ calculatePreis(basisPreis : Double) : Double

"STANDARD"	"PREMIUM"	typ
Rückgabe: basisPreis	Rückgabe: basisPreis * 0.95	Rückgabe: basisPreis * 0.9

Da zukünftig weitere Besuchertypen geplant sind, rät Ihnen ein erfahrener Kollege, diese Redundanz durch Polymorphie aufzulösen, um die Wartbarkeit des Softwareprodukts zu verbessern.

ba) Erläutern Sie den Begriff Polymorphie.

4 Punkte

bb) Erstellen Sie ein UML-Klassendiagramm für den polymorphen Ansatz ohne redundante Auswahlstrukturen. 9 Punkte Korrekturrand

Hinweise:

- Machen Sie dazu die Klasse *Besucher* abstrakt.
- Entfernen Sie die Instanzvariable *typ*.
- Verwenden Sie Vererbung und Überschreibung von abstrakten Methoden.
- Modellieren Sie in der Klasse *Besucher* eine statische Fabrikmethode *createBesucher*, die je nach Besucher-Typ die entsprechende Besucherinstanz zurückliefern soll.