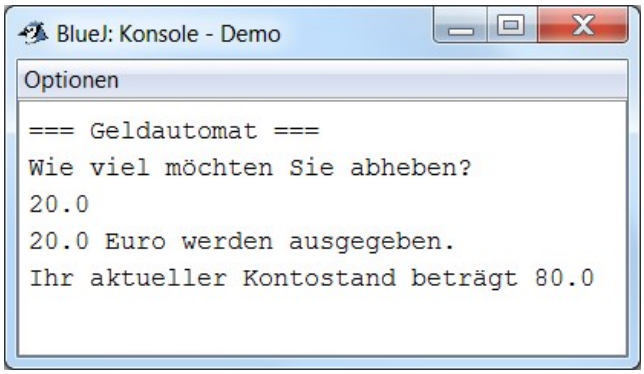
Die **Konsole** ist ein **Textfenster**, mit dem Programme gesteuert werden können.
Über die Konsole interagieren Programm und Benutzer nur über Text. Das heißt, das Programm kann einen Text schreiben (**Ausgabe**, output), und der Benutzer kann reagieren, indem er Text eintippt (**Eingabe**, input). Das Programm liest die Eingabe des Benutzers ein und reagiert darauf wieder durch Ausgabe von Text, und so weiter.

Synonyme für Konsole sind „Terminal“, „Kommandozeile“, „Eingabeaufforderung“, „Shell“.

Mit der Konsole kann man **Benutzerschnittstellen** (engl. user interfaces, UI) programmieren, das heißt den Teil eines Programms, der mit dem Benutzer interagiert.

Betrachten wir als Beispiel einen einfachen **Geldautomaten**: Er bittet den Benutzer, einen Betrag einzutippen, der vom Konto abgehoben wird:

- Das Programm macht eine **Ausgabe** (schreibt Text und wartet)
- Der Benutzer macht eine **Eingabe** (tippt einen Text und drückt Enter)
  Dabei wird der Kontostand verändert.
- Das Programm macht eine **Ausgabe** (mit dem Ergebnis)

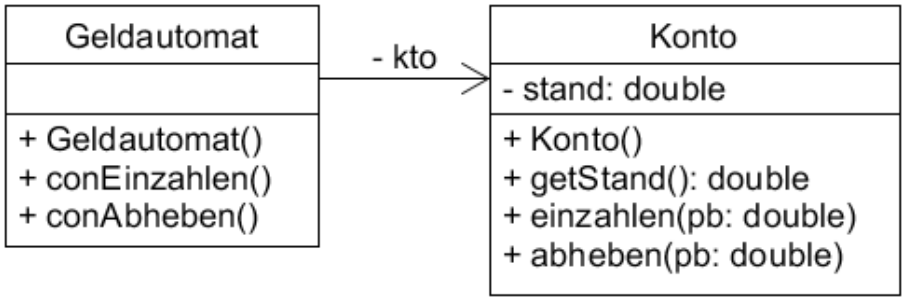


Trennung von Benutzerschnittstelle und Daten

Wie auch in den bisherigen Beispielen ist das Programm in (mindestens) zwei Klassen aufgeteilt. Objekte der einen Klasse (hier: Konto) enthalten die **Daten**, also die Kontostände, und Methoden, mit denen man die Daten abfragen und verändern kann.

Von einer anderen Klasse (hier: Geldautomat) wird das Hauptobjekt des Programms erstellt.
Diese Klasse enthält Methoden, die die **Benutzerschnittstelle** implementieren. Hier wird die Konsole dafür verwendet, daher sind die Methoden hier z.B. mit „conEinzahlen()“ benannt, also der Dialog per Konsole mit dem Kunden beim Einzahlen von Bargeld in den Geldautomaten.

Ein **Vorteil** der Trennung von Benutzerschnittstelle und Daten ist, dass man nicht nur eine, sondern unterschiedliche Benutzerschnittstellen anbieten kann (z.B. eine für den PC und eine fürs Handy). Für eine andere Benutzerschnittstelle könnte man einfach eine andere Klasse implementieren, und die Klasse für die Daten (hier: Konto) unverändert lassen.



Aufgabe 1

Öffne das bereitgestellte BlueJ-Projekt zum Geldautomaten.
Lies den Quellcode und beantworte die folgenden Fragen.
Die Dokumentation auf Seite 4 lies erst im Anschluss, oder bei Verständnisschwierigkeiten.

a) Beschreibe, was während der Ausführung der Methode **conAbheben**() der Klasse **Geldautomat** geschieht: Was sieht der Benutzer, was soll der Benutzer tun, was passiert mit den Objekten während des Ablaufs?

b) Mit welcher Methode gibt man generell einen Text auf der Konsole aus?

c) Wenn der Benutzer eine Dezimalzahl auf der Konsole eingeben soll:
Mit welcher Methode liest das Programm diese Eingabe ein?
Wie kann die eingegebene Zahl im Programm weiter verarbeitet werden?

d) Wie lautet wohl der entsprechende Befehl für eine ganze Zahl?
Und wie für einen Text?

Aufgabe 2

a) Zur Wiederholung: Erweitere die Methoden der Klasse **Konto**, so dass geprüft wird, ob die Parameter sinnvolle Werte enthalten (falls ein Parameter einen fehlerhaften Wert enthält, wird der Kontostand nicht geändert).

b) Erweitere die Methode **conAbheben**() der Klasse **Geldautomat**:
Nachdem der Benutzer eingegeben hat, wie viel er abheben möchte, soll das Programm prüfen, ob der eingegebene Wert positiv ist (falls nicht, wird eine Fehlermeldung ausgegeben).
Dann wird geprüft, ob genügend Geld auf dem Konto vorhanden ist, und nur in diesem Fall wird der Betrag auch abgehoben. Andernfalls gibt es eine Fehlermeldung.
Am Ende wird dann (in jedem Fall) der aktuelle Kontostand ausgegeben.

Hinweis: Beachte die **Hilfestellung** zur Abfrage mehrerer Bedingungen auf Seite 5.

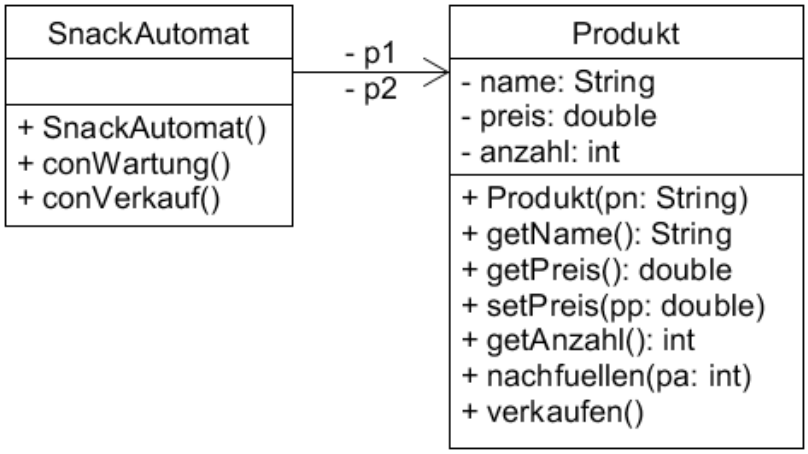
c) Füge die Methode **conEinzahlen**() zur Klasse **Geldautomat** hinzu:
Auch hier soll der Benutzer einen Betrag eingeben.
Es wird geprüft, ob der Betrag positiv ist (falls nicht: Fehlermeldung).
Dann wird der Betrag auf das Konto eingezahlt, und der neue Kontostand wird ausgegeben.

Aufgabe 3

Ein Snackautomat bietet Produkte, z.B. Getränke, Schokolade, Chips usw. an.
Das Programm, das den Automaten steuert, besteht aus zwei Klassen:

Objekte der Klasse **Produkt** stehen für ein Produkt und speichern dessen Namen, wie viel eine Einheit des Produkts kostet, und die Anzahl von Einheiten, die der Automat aktuell vorrätig hat. Der Name des Produkts wird im Konstruktor gesetzt, für den Preis gibt es get- und set-Methode. Die Methode nachfüllen() erhöht die Anzahl der Einheiten des Produkts, verkaufen() verringert sie um 1 (nur, falls noch Produkte vorrätig sind).

Die Klasse **SnackAutomat** enthält die Benutzerschnittstelle. Es gibt zwei verschiedene Modi:
Die Methode conVerkauf() führt einen Kunden durch die Wahl eines Produkts und das Bezahlen.
Die Methode conWartung() führt einen Mitarbeiter der Firma durch das Nachfüllen der Produkte.



Verwende die BlueJ-Vorlage für Aufgabe 3. Die Klasse Produkt ist schon fertig implementiert, du brauchst also nur die Klasse **Snackautomat** zu bearbeiten.

a) Implementiere den **Konstruktor** SnackAutomat():
Er erzeugt die Produkte p1 und p2 und gibt ihnen dabei je einen Namen.

b) Implementiere die Methode **conWartung**():
Sie fragt den Mitarbeiter für beide Produkte, wie viel nachgefüllt werden soll, und nach den neuen Preisen.

c) Implementiere die Methode **conVerkauf**():
Sie gibt zunächst aus, welche Produkte der Automat verkauft und wie viel eine Einheit jeweils kostet. Sie fragt, welches Produkt der Kunde möchte (der Kunde kann durch Eingabe einer Nummer, also 1 oder 2, reagieren). Falls noch Einheiten des Produkts vorhanden sind, wird der Benutzer gebeten, den Geldbetrag einzutippen, den er einwirft (in Wirklichkeit würde das Programm natürlich die Münzen zählen, aber wir simulieren das hier einfach mit der Konsole). Das Programm gibt eine Einheit des Produkts ab und gibt aus, wie viel Wechselgeld der Kunde bekommt.
Falls das Produkt ausverkauft ist, wird stattdessen eine entsprechende Meldung ausgegeben.

Dokumentation

`import console.*;`	Importiert die Klasse Console.

Klasse Console

Es ist nicht nötig, ein **Objekt** der Klasse Console zu erzeugen: Es braucht also keine Anweisung wie `con = new Console().` Die Methoden können direkt über die Klasse Console ausgeführt werden, z.B. `Console.println(…).`	
`void clear()`	Löscht den bisherigen Inhalt der Konsole.
`void print(String ptext)`	Schreibt einen Text auf die Konsole (ohne Zeilenumbruch)
`void println(String ptext)`	„Print Line“: Schreibt einen Text, und der Cursor geht anschließend in die nächste Zeile.
`int readInt()`	Wartet, bis der Benutzer eine ganze Zahl eingegeben und Enter gedrückt hat. Die eingegebene Zahl wird zurückgegeben.
`double readDouble()`	Wie readInt(), nur für Dezimalzahlen
`char readChar()`	Wie readInt(), nur für einen einzelnen Buchstaben
`String readString()`	Wie readInt(), nur für ein einzelnes Wort
`String readln()`	„Read Line“: Liest eine ganze Zeile (mit Leerzeichen) als String ein, bis zum Zeilenumbruch

Beispiel

```
01  public void quadratBerechnen()
02  {
03     int z, q;
04     Console.println("Guten Tag");
05     Console.print  ("Geben Sie eine Zahl ein: ");
06     z = Console.readInt();
07     q = z * z;
08     Console.println("Das Quadrat der Zahl ist " + q);
09  }
```

In Zeile 05 wartet der Cursor am Ende der Zeile auf die Eingabe des Benutzers.
Daher steht im Text auch noch ein Leerzeichen hinter dem Doppelpunkt.

Zeile 06 wartet dann solange, bis der Benutzer eine Zahl getippt und Enter gedrückt hat.

In Zeile 08 wird der Text „Das Quadrat der Zahl ist“ ausgegeben, ebenfalls mit einem Leerzeichen am Ende. Dahinter wird der Wert der Variablen q ausgegeben, der in Zeile 07 berechnet wurde.
Auf diese Weise kann man dem Benutzer Ergebnisse von Rechnungen oder Daten wie den Kontostand mitteilen. Das + bedeutet hier also nicht „Addition“, sondern hintereinander hängen mehrerer Texte bzw. Variableninhalte. Die Variable darf nicht innerhalb der Gänsefüßchen stehen.

Verkettung von if-else

In Konsolenanwendungen (oder allgemein in Benutzerschnittstellen) muss oft eine Reihe von Fällen behandelt werden, um z.B. mögliche Fehler auszuschließen.
Meist muss das Programm auf jeden dieser Fälle unterschiedlich reagieren, daher genügt ein einfaches if (Fall 1 oder Fall 2 oder Fall 3) nicht.

Eine Reihe von Fällen kann man mit einer Kette von if – else Anweisungen abhandeln.

Beispiel: Der Benutzer soll eingeben, wie viel Geld er abheben möchte.

```
if ( <Betrag negativ> )
{
   <Fehlermeldung: nicht möglich>
}
else if ( <Betrag mehr als vorhanden> )
{
   <Fehlermeldung: Kontostand reicht nicht aus>
}
else
{
   <Betrag vom Konto abziehen>
   <Meldung: erfolgreich>
}
```

Die Anweisungen im letzten „else“-Teil werden nur ausgeführt, wenn die vorigen Bedingungen alle nicht zutrafen, d.h. wenn alle möglichen Fehler ausgeschlossen wurden.