Temat do przemyślenia Co *konkretnego* widzisz w swoim otoczeniu? Co możesz z takimi rzeczami zrobić? Czy to samo możesz zrobić z czymś innym? Jakie jest wspólne zachowanie (czyli interfejs), jakie one wykazują?

24.1 Typ interfejsowy

Typy definiują przechowywane wartości: liczby całkowite, łańcuchy itd. Z typem interfejsowym jest inaczej. Interfejsy opisują, co dany typ może zrobić, a nie jego wartości.

Metody opisują zachowanie typu, a więc interfejsy deklaruje się, wskazując zbiór metod, które dany typ musi implementować (mówimy, że dany typ musi „spełniać" jakiś interfejs). Poniższy listing deklaruje zmienną typu interfejsowego.

Listing 24.1 Zbiór metod – talk.go

```
var t interface {
    talk() string
}
```

Zmienna `t` może zawierać dowolną wartość dowolnego typu, który *spełni* tak zdefiniowany interfejs. Dokładniej, dany typ spełni wymagania tego interfejsu, jeśli zadeklaruje metodę o nazwie `talk` bez argumentów, zwracającą łańcuch znaków.

Poniższa lista deklaruje dwa typy, które spełniają te wymagania.

Listing 24.2 Spełnianie wymagań interfejsu – talk.go

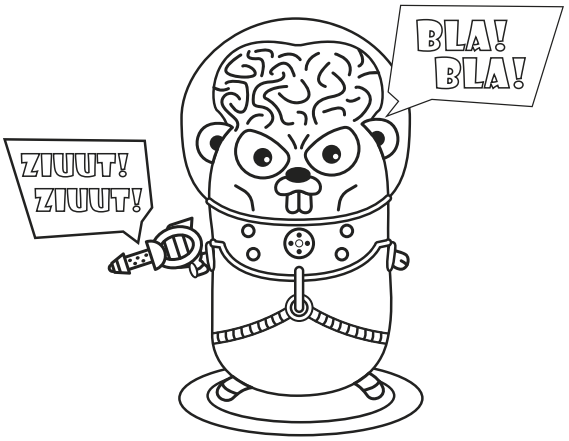
```
type martian struct{}          // marsjanin
func (m martian) talk() string  {
    return "bla bla"
}
type laser int                 // laser
func (l laser) talk() string {
    return strings.Repeat("ziuut", int(l))
}
```

Chociaż `martian` jest strukturą bez pól, a `laser` jest liczbą całkowitą, to oba typy udostępniają metodę `talk` i dlatego mogą zostać przypisane do zmiennej `t`, co pokazuje poniższy listing.

Listing 24.3 Polimorfizm – talk.go

```
var t interface {
    talk() string
}
t = martian{}
fmt.Println(t.talk()) ←—— Wyświetli: bla bla
t = laser(3)
fmt.Println(t.talk()) ←—— Wyświetli: ziuut ziuut ziuut
```

Dziwna, bezkształtna zmienna `t` może raz przyjąć postać marsjanina, a innym razem `lasera`. Informatycy twierdzą, że interfejsy zapewniają *polimorfizm*, czyli „zdolność przyjmowania wielu postaci".



UWAGA W przeciwieństwie do Javy, w języku Go deklaracje `martian` i `laser` w żaden sposób nie informują, że implementują interfejs. Korzyści z tego zostaną omówione w dalszej części tej lekcji.

Zazwyczaj interfejsy deklaruje się jako typy nazwane, które można ponownie wykorzystać. Zaleca się tu użycie konwencji nazewniczej dla typów interfejsów, która polega na dodawaniu przyrostka *-er* (w języku polskim byłby to -arz): a więc *baker* (piekarz), czyli ktoś, kto umie piec (*bake*). Na poniższym listingu mamy interfejs talker, który wymaga metody talk.

Listing 24.4 Typ talker – shout.go

```
type talker interface {
    talk() string
}
```