CSI2520: PARADIGMES DE PROGRAMMATION
Tutorat 2: méthodes et interface

Certains de ces exercices sont tires de “The Little Go Book” https://www.openmymind.net/The-Little-Go-Book/

Exercice 1 : Les méthodes

1. Créer une structure rect qui stocke la hauteur et largeur d’un rectangle

2. Créer une fonction area qui calcule l’aire d’un rectangle

3. Créer une fonction perim qui calcule le périmètre d’un rectangle

4. Créer une fonction main qui utilise les fonctions et la structure ci-dessus pour afficher l’aire et le périmètre d’un rectangle

```
package main
import "fmt"
type rect struct {
    width, height int
}
func (r *rect) area() int {
    return r.width * r.height
}
func (r rect) perim() int {
    return 2*r.width + 2*r.height
}
func main() {
    r := rect{width: 10, height: 5}
    fmt.Println("area: ", r.area())
    fmt.Println("perim:", r.perim())
}
```

Exercice 2 : Les interfaces

1. Définir une interface pour les formes géométriques qui contient les méthodes area et périmètre

2. Définir une fonction *measure* qui prend comme paramètre une interface géométrique et affiche l'aire et le périmètre de la forme géométrique.

Définir une fonction *main* qui utilise la fonction *measure* pour afficher l'aire et le périmètre d'un carré et d'un cercle.

```
package main

import (
    "fmt"
    "math"
)

// define interface shape: has two methods
type shape interface {
    name() string
    area() float64
    perimeter() float64
}

// declare types rectangle, circle
type rect struct {
    s string
    height, width float64
}

type tri struct {
    s string
    height, base float64
}

type circ struct {
    s string
    radius float64
}

// name getters
func (r rect) name() string { return r.s }
func (t tri) name() string { return t.s }
func (c circ) name() string { return c.s }

// method invoked by ex: r1.area()
func (r rect) area() float64 {
    return r.height * r.width
}
```

```
// r1.perimeter()
func (r rect) perimeter() float64 {
    return 2*r.height + 2*r.width
}

// pass r by pointer so function can modify it
// note that double() is not required by shape interface
// but that doesn't matter
func (r *rect) double() {
    r.height *= 2
    r.width *= 2
}

func (t tri) area() float64 {
    return 1. / 2. * t.height * t.base
}

func (t tri) perimeter() float64 {
    // assumes isoleces triangle
    c := math.Sqrt(math.Pow(t.height, 2) + math.Pow(1./2.*t.base, 2))
    return t.base + 2*c
}

func (c circ) area() float64 {
    return math.Pi * math.Pow(c.radius, 2)
}

func (c circ) perimeter() float64 {
    // adding period to 2 to specify float
    return 2. * math.Pi * c.radius
}

func measure(s shape) {
    fmt.Printf("Shape: %s\n", s.name())
    fmt.Printf("Area: %f\n", s.area())
    fmt.Printf("Perimeter: %f\n\n", s.perimeter())
}

func main() {
    // make a slice of shapes of length 0
    shapes := make([]shape, 0)
    //shapes := []shape{} // alternate syntax using type deduction

    r1 := rect{"rectangle 1", 10, 5}
    shapes = append(shapes, r1)

    r2 := r1 // copy r1
    r2.double()
    r2.s = "rectangle 2"
    // modify member
    //measure(r1)
```

```
    //measure(r2)
    // r1 is unchanged
    shapes = append(shapes, r2)

    t1 := tri{"triangle 1", 10, 5}
    shapes = append(shapes, t1)

    c1 := circ{"circle 1", 1}
    shapes = append(shapes, c1)

    for _, s := range shapes {
        // measure takes argument of interface type shape
        // so works on rect, circ, tri
        measure(s)
    }
}
```

Exercice 3 : Composition avec des types embarqués

La composition en Go est un outil de réutilisation de code. Il est utile lorsqu'un veut bâtir un nouveau type en enrichissant un type existant. Cela s'apparente à de l'héritage mais dans un cas ou les méthodes réutilisées ne sont pas modifiées.

```
package main

import "fmt"

type Personne struct {
  Nom string
  Prenom string
}

func (p *Personne) Bonjour() {
  fmt.Printf("Bonjour, Je suis %s %s\n", p.Prenom, p.Nom)
}

type Employé struct {
  Personne
  Salaire float32
}


func main() {

  p1 := Personne{"Cesar", "Jules"}
  p1.Bonjour()

  p2 := &Employé{Personne{"Gates","Bill"}, 15.25}
  p2.Bonjour()
}
```