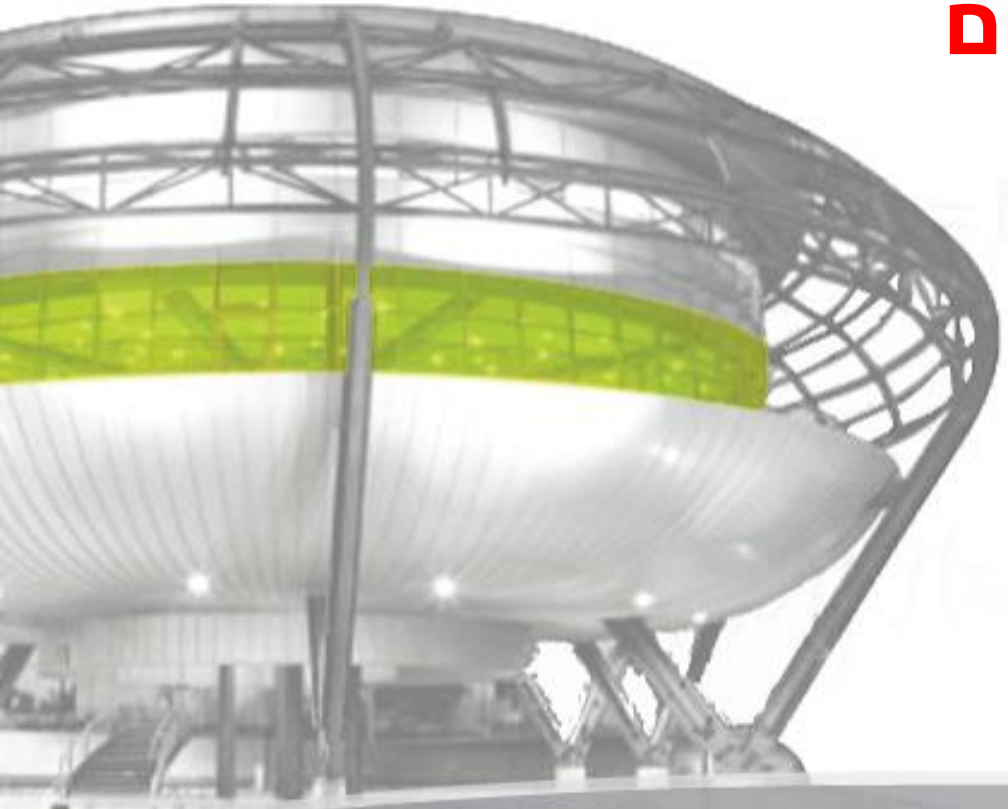
תכנות מונחה עצמים

סמסטר קיץ תשעט

מעבדה 4



sce
המכללה האקדמית להנדסה סמי שמעון

תתחבר לרעיונות גדולים
באר שבע | אשדוד | www.sce.ac.il | מהנדס

Composition - reminder

- כאשר מחלקה מכילה מחלקה אחרת, אתחול האובייקטים המוכלים מתבצע לפני אתחול האובייקט המכיל.
- אתחול אובייקט מתבצע ב-Ctor, לכן יש מעבר ב-Ctor של האובייקט המוכל לפני הכניסה לגוף ה-Ctor של האובייקט המכיל.
- אתחול זה מבוצע ב-init line, המבוצעת לפני הכניזה לגוף ה-Ctor.
- במקרה והאובייקט המוכל לא אותחל במפורש בשורת האתחול, יהיה ניסיון לאתחלו דרך ה-default Ctor, במידה ואינו קיים תתקבל שגיאת קומפילציה.
- סדר הריסת האובייקטים הפוך לסדר היצירה, כלומר קודם נהרס האובייקט המכיל ורק אז האובייקט המוכל.

Composition and copy Ctor

- ה-copy Ctor שמקבלים במתנה מהקומפיילר מפעיל את ה- copy Ctor של האובייקט המוכל לפני כניסה לגוף ה-Ctor של האובייקט המכיל.
- במידה ודורסים את ה-copy Ctor שמקבלים מהקומפיילר יש לזכור להפעיל Ctor כלשהו של האובייקט המוכל אחרת יופעל ה-default Ctor של האובייקט המוכל.

Example

```cpp
#ifndef __POINT_H
#define __POINT_H
#include <iostream>
using namespace std;
class Point {
public:
   void setX(int m_x);
   void setY(int m_y);
   int getX() { return x; }
   int getY() { return y; }
   Point();
   Point(int m_x, int m_y);
   Point(const Point& other);
   ~Point();
   inline void print() {
      cout << "x = " << x << ", y = " << y << endl;
   }
private:
   int x, y;
};
#endif //__POINT_H
```

```cpp
#include "Point.h"
void Point::setX(int m_x) { x = m_x; }
void Point::setY(int m_y) { y = m_y; }
Point::Point() : Point(0, 0) {
}
Point::Point(int m_x, int m_y) {
   x = m_x;
   y = m_y;
}
Point::Point(const Point& other) {
   x = other.x;
   y = other.y;
}
Point::~Point() {
   cout << "Deleting a point" << endl;
}
```

Example

```
#include "Point.h"
#ifndef __CIRCLE_H
#define __CIRCLE_H
class Circle {
public:
   void setRadius(double m_radius) { radius = m_radius; }
   void setCenter(Point m_center);
   double getRadius() { return radius; }
   Point getCenter() const { return center; }
   Circle();
   Circle(Point m_center, double m_radius);
   Circle(const Circle& other);
   ~Circle();
   void print();
private:
   double radius;
   Point center;
};
#endif //__CIRCLE_H
```

Example

```
#include "Circle.h"
void Circle::setCenter(Point m_center) {
   center = Point(m_center.getX(), m_center.getY());
}
Circle::Circle() : center(0, 0) {
   radius = 0;
}
Circle::Circle(Point m_center, double m_radius): center(m_center) {
   radius = m_radius;
}
Circle::Circle(const Circle& other) : center(other.getCenter()) {
   radius = other.radius;
}
Circle::~Circle() {
cout << "Deleting a circle" << endl;
}
void Circle::print() {
   cout << "Circle radius: " << radius << ", circle center: ";
   center.print();
}
```

גישה לאובייקט בתוך מתודה

- פונקציה גלובלית – יכולה לעבוד **רק** עם פרמטרים שהועברו אליה, או משתנים מקומיים שלה.
- מתודה פועלת גם על שדות(data members) של המחלקה.
- בתוך כל מתודה של מחלקה אנו פועלים על אובייקט ספציפי(האובייקט שהפעיל את המתודה)
- שם האובייקט הספציפי אינו ידוע בתוך המתודה.
- תיתכן בעיה כאשר המתודה תרצה להתייחס לאובייקט בכללותו, ולא רק לאחת מתכונותיו מאחר ואינה יודעת את שמו.
- הפתרון: שימוש במצביע **this**, שזו מילה שמורה בשפה שמשמעותה פניה לאובייקט בהפעיל את המתודה.

המצביע this

- כאשר קוראים למתודה מסוימת בנוסף לפרמטרים הרגילים שלה, נשלח פרמטר נוסף(נסתר) הנקרא this.
- **this הוא מצביע לאובייקט שממנו נקראת המתודה.**
- **זה מצביע מסוג const (לא ניתן לשנות אותו אבל ניתן לשנות את האובייקט שעליו הוא מצביע).**
- בכל גישה לשדה פנימי של המחלקה, הקומפיילר מוסיף **this->**.

המצביע this

- this היא מילה שמורה שמייצגת מצביע לאובייקט שהפעיל את המתודה וניתן להשתמש בזה עבור:
- רישום מפורש של שם משתנה במקרה של פרמטר עם אותו שם – בעזרת המצביע this נוכל לתת שמות משמעותיים למשתנים במתודות.
- שליחת האובייקט לפונקציה אחרת.
- החזרת האובייקט כערך החזרה.

המצביע this

- this היא מילה שמורה שמייצגת מצביע לאובייקט שהפעיל את המתודה וניתן להשתמש בזה עבור:
- רישום מפורש של שם משתנה במקרה של פרמטר עם אותו שם.
- שליחת האובייקט לפונקציה אחרת.
- החזרת האובייקט כערך החזרה.

Example

```
#ifndef __POINT_H
#define __POINT_H
#include <iostream>
using namespace std;
class Point {
public:
   void setX(int x);
   void setY(int y);
   int getX() { return x; }
   int getY() { return y; }
   Point();
   Point(int x, int y);
   Point(const Point& other);
   ~Point();
   inline void print() {
      cout << "x = " << x << ", y = " << y << endl;
   }
private:
   int x, y;
};
#endif //__POINT_H
```

```
#include "Point.h"
void Point::setX(int x) { this->x = x; }
void Point::setY(int y) { this->y = y; }
Point::Point() : Point(0, 0) {
}
Point::Point(int x, int y) {
   this->x = x;
   this->y = y;
}
Point::Point(const Point& other) {
   x = other.x;
   y = other.y;
}
Point::~Point() {
   cout << "Deleting a point" << endl;
}
```

המצביע this

במקרה של פרמטר עם שם זהה לתכונה, לפרמטר יש עדיפות בתוך השיטה, ולכן צריך להקפיד לקרוא לתכונה דרך המצביע this

```
void Point::setX(int x) {
    this->x = x;
}
```

פניה לתכונה

פניה לפרמטר

Conclusion

- כדי לפנות לתכונה או למתודה של האובייקט נשתמש ב- ".".
- כאשר המשתנה הוא מצביע, הפניה היא באמצעות <-
- כדי לפנות לאובייקט שהפעיל את המתודה נשתמש ב-this