# לקוחות, ספקים ומנשקים
## בשפת C



## לקוח וספק במערכת תוכנה

- ספק (supplier) – הוא מי שקוראים לו (לפעמים נקרא גם שרת, server)
- לקוח (client) הוא מי שקרא לספק או מי שמשתמש בו (לפעמים נקרא גם משתמש, user)
- דוגמא:

```
void do_something()
{
  // doing...
}

main()
{
  do_something();
}
```


• בדוגמא זו הפונקציה main היא **לקוחה** של הפונקציה do_something()

• do_something היא **ספקית** של main



## לקוח וספק במערכת תוכנה

- הספק והלקוח עשויים להיכתב בזמנים שונים, במקומות שונים וע"י אנשים שונים ואז כמובן לא יופיעו באותו קובץ

supplier.c

```
void do_something()
{
  // doing...
}
```

client.c

```
main()
{
  do_something();
}
```



## לקוח וספק במערכת תוכנה

- מנקודת מבטו של הלקוח קוד המקור של הספק עשוי לא להיות זמין כלל

supplier.o

```
#@!!!>>??%^#&@
```

client.c

```
main()
{
      do_something(???);
}
```



## לקוח וספק במערכת תוכנה

- כדי לתקשר בין הספק והלקוח עליהם להגדיר מנשק (interface, ממשק) בינהם

**בצד הלקוח**

```
#include "supplier.h"
main()
{
    do_something(???);
}
```

client.c

**בצד הספק**

supplier.h

```
void do_something();
```

```
#include "supplier.h"
void do_something()
{
      // doing...
}
```

supplier.c



## לקוח וספק במערכת תוכנה

- לאחר תהליך ההידור מקבל הלקוח גם קובץ בינארי (o. או OBJ.) וגם קובץ כותרות (h.)

**בצד הלקוח**

supplier.o

```
#@!!!>>??%^#&@
```

supplier.h

```
void do_something();
```

```
#include "supplier.h"
main()
{
      do_something();
}
```

client.c

**בצד הספק**

supplier.o

```
#@!!!>>??%^#&@
```

supplier.h

```
void do_something();
```

```
#include "supplier.h"
void do_something()
{
      // doing...
}
```

supplier.c

## מנשק תחילה

- בתהליך פיתוח תוכנה תקין, כתיבת המנשק תעשה בתחילת תהליך הפיתוח
- כל מודול מגדיר מהם השרותים שלהם הוא זקוק ממודולים אחרים ע"י ניסוח מנשק רצוי
- מנשק זה מהווה בסיס לכתיבת הקוד הן בצד הספק, שיממש את הפונקציות הדרושות והן בצד הלקוח, שמשתמש בפונקציות (קורא להן) ללא תלות במימוש שלהן

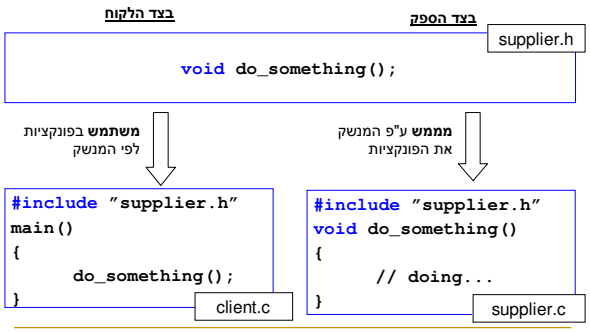


## מנשקים

- מטרת המנשק היא למזער את התלות בין חלקים שונים של המערכת
- מנשקים זעירים יוצרים מערכת:
  - קלה יותר להבנה
  - בעלת הסתרת מידע טובה יותר
  - קלה לתחזוקה ושינויים
  - מתקמפלת במהירות



## לקוח וספק במערכת תוכנה






## ריבוי מנשקים

- מתכנתים ומהדרים נוקטים בגישה הקונסרבטיבית (המוצדקת!) – אם שרות כלשהו זמין ללקוח כלשהו יש להניח כי הלקוח תלוי בקטע קוד זה
- כלומר יש צורך להגדיר מנשקים נפרדים (לא בהכרח זרים) לממשים ולמשתמשים (ניתן אפילו לחשוב על מנשקים שונים למשתמשים שונים)



## מנשקים ו Java

- ניתוח והבנה של מערכת תוכנה במונחי ספק-לקוח והמנשקים בינהם היא אבן יסוד בכתיבת תוכנה מודרנית
- בשפת C המנשק אינו מתבטא **בשפת התכנות**, ה pre-processor הוא זה שיוצר אותו, ועל המתכנת לאכוף את עיקביותו



## מנשקים ו Java

- בשפת Java הפך המנשק לרכיב בשפה – המהדר אוכף את עקביותו
- עקב כך, אין ב Java קובצי כותרת כלל
- כמו כן, אין צורך להצהיר על פונקציות לפני השימוש בהן
- נדון במנשקים בהרחבה בהמשך הקורס...

# חושבים עצמים

הסבה של POINT מ- struct בשפת C
ל- class בשפת Java



## מ C ל- Java עם POINT

בשפת C

```
typedef struct POINT
{
   float x,y;
} POINT;

float rho(POINT *this)
{
   return sqrt((this->x * this->x) +
                          (this->y * this->y));
}

main()
{
   float r;
   POINT p = {3,4};
   r = rho(&p);
}
```



## מ C ל- Java עם POINT

בשפת C

```
typedef struct POINT
{
   float x,y;
} POINT;
```

בשפת Java

```
class POINT
{
        float x,y;
}
```

typedef struct
הפך להיות class



## מ C ל- Java עם POINT

בשפת C

```
float rho(POINT *this)
{
   return sqrt((this->x * this->x) +
                          (this->y * this->y));
}
```

בשפת Java

```
class POINT
{
        float x,y;
        float rho(POINT *this)
        {
                return sqrt((this->x * this->x) +
                                  (this->y * this->y));
        }
}
```

הגדרת הפונקציה נכנסה לתוך הגדרת המחלקה (הפונקציה הפכה למתודה)



## מ C ל- Java עם POINT

בשפת C

```
float rho(POINT *this)
{
   return sqrt((this->x * this->x) +
                          (this->y * this->y));
}
```

בשפת Java

```
class Point{

        float x,y;

        float rho(Point *this){
                return sqrt((this->x * this->x) +
                                  (this->y * this->y));
        }
}
```

מוסכמה סגנונית – פתיחת בלוק מתחילה שורה קודם

מוסכמה סגנונית – שמות מחלקות בשיטת הגמל

## מ C ל- Java עם POINT

בשפת C

```
float rho(POINT *this)
{
   return sqrt((this->x * this->x) +
                          (this->y * this->y));
}
```

בשפת Java

```
class Point{

        float x,y;

        float rho(Point this){
                return sqrt((this.x * this.x) +
                                  (this.y * this.y));
        }
}
```

אין ב Java הבדל בין התייחסות לעצם ובין מצביע לעצם –
יש לוותר על הכוכבית והחץ

## מ C ל- Java עם POINT

בשפת C

```
float rho(POINT *this)
{
   return sqrt((this->x * this->x) +
                          (this->y * this->y));
}
```

בשפת Java

```
class Point{

       float x,y;

       float rho(){
              return sqrt(x*x + y*y);
       }
}
```

אבל מומלץ לוותר על `this`



## מ C ל- Java עם POINT

בשפת C

```
float rho(POINT *this)
{
   return sqrt((this->x * this->x) +
                          (this->y * this->y));
}
```

בשפת Java

```
class Point{

       float x,y;

       float rho(){
              return sqrt((this.x * this.x) +
                                   (this.y * this.y));
       }
}
```

המתודה לא מקבלת את `Point` בתור ארגומנט (כי היא מוגדרת בתוכה ולכן ברור שהיא פועלת עליה). עדיין מותר להשתמש ב `this` אם רוצים

## מ C ל- Java עם POINT

בשפת C

```
float rho(POINT *this)
{
   return sqrt((this->x * this->x) +
                          (this->y * this->y));
}
```

בשפת Java

```
class Point{

       private float x,y;

       public float rho()  {
              return sqrt(x*x + y*y);
       }

}
```

כדי למנוע את הגישה לשדות `x` ו- `y` מחוץ למחלקה יש להוסיף את המציין `private` לפני הגדרת השדות

## מ C ל- Java עם POINT

בשפת C

```
float rho(POINT *this)
{
   return sqrt((this->x * this->x) +
                          (this->y * this->y));
}
```

בשפת Java

```
class Point{

       float x,y;

       public float rho()  {
              return sqrt(x*x + y*y);
       }

}
```

כדי שאפשר יהיה לקרוא למתודה מחוץ למחלקה יש להוסיף את המציין `public` לפני הגדרת המתודה

## מ C ל- Java עם POINT

בשפת C

```
float rho(POINT *this)
{
   return sqrt((this->x * this->x) +
                          (this->y * this->y));
}
```

בשפת Java

```
class Point{

       private float x,y;

       public float rho()  {
              return Math.sqrt(x*x + y*y);
       }

}
```

אין ב `Java` פונקציות גלובליות. כל מתודה (פונקציה) מוגדרת במחלקה כלשהי. שם המחלקה הוא חלק משמה המלא של הפונקציה

## מ C ל- Java עם POINT

בשפת C

```
main()
{
   float r;
   POINT p = {3,4};
   r = rho(&p);
}
```

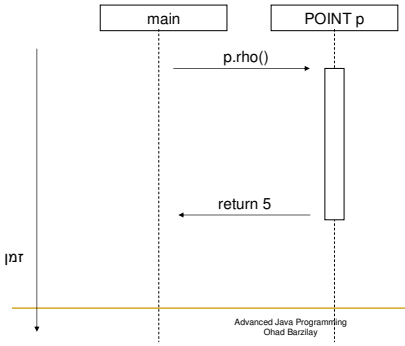
בשפת C אנו מסתכלים על תוכנית כעל אוסף של פונקציות שמקבלות כארגומנטים מבני נתונים

ב Java אנו מסתכלים על תוכנית כעל סדרה של הודעות \ בקשות המועברות בין אובייקטים ובהם הם מבקשים זה מזה שרותים שונים

בשפת Java

```
public static void main(String[] args) {
   float r;
   Point p = new Point(3,4);
   r = p.rho();
}
```

## מודל העברת ההודעות






## Hello Object Oriented World



## class Greeting (supplier)

```
public class Greeting {
  public void greet() {
    System.out.println("hi");
  }
}
```

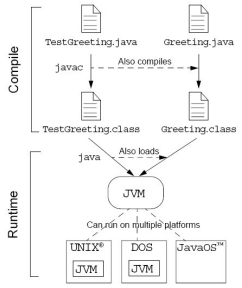


## class TestGreeting (Client)

```
/** Sample "Hello World" User */
public class TestGreeting{
   public static void main (String[] args) {
      Greeting hello = new Greeting();
      hello.greet();
   }
}
```



## Building the Application






## Homework (no submission)

- Download JDK from http://java.sun.com/j2se/1.5.0/download.jsp and install it on your PC
- Download Eclipse from http://www.eclipse.org/downloads/ and unzip it (no installation needed)
- Read eclipse documentation from the course web site: http://www.cs.tau.ac.il/~ohadbr/advJava
- Compile and Run **`TestGreeting`** application