ABSTRACT CLASSES

Polymorphism - Abstract Classes

- ישנם מקרים (במיוחד כשרוצים לממש מבני נתונים), שבהם אין לנו שום כוונה באמת לייצר אובייקט ממחלקת הבסיס.
- במקרים כאלו אנו מעוניינים ל:
  - **למנוע** מהמתכנת את האפשרות **לייצר אובייקט** מסוג מחלקת הבסיס.
  - **להימנע ממימוש** חלק מ**המתודות** הווירטואליות של מחלקת הבסיס (הבנים יכולים לשכתב).
  - **להכריח** את המחלקות הנגזרות לשכתב את המתודות שלא מומשו במחלקת הבסיס.
- הפתרון הוא מתודות ווירטואליות טהורות (**Pure Functions**) ומחלקות אבסטרקטיות (**Abstract Classes.**).

Polymorphism - Abstract Classes cont

- הפתרון הוא מתודות ווירטואליות טהורות (**Pure Functions**) ומחלקות אבסטרקטיות (**Abstract Classes.**):
  - מתודה ווירטואלית טהורה היא מתודה וירטואלית שמוצהרת במחלקת הבסיס אך ורק לשם הוספת למימוש והכרחת הצאצאים שלה לממש אותה.
  - לשם הגדרתה – יש להוסיף 0= להגדרה (prototype) של המתודה בקובץ ה-H.
  - המתודה **לא תמומש** (למעט Dtor, פרטים בהמשך).
  - מחלקה עם מתודות pure virtual (לפחות אחת) היא אוטומטית – מחלקה אבסטרקטית! -> לא ניתן לייצר אובייקטים מסוג מחלקה זאת (שגיאת קומפילציה).

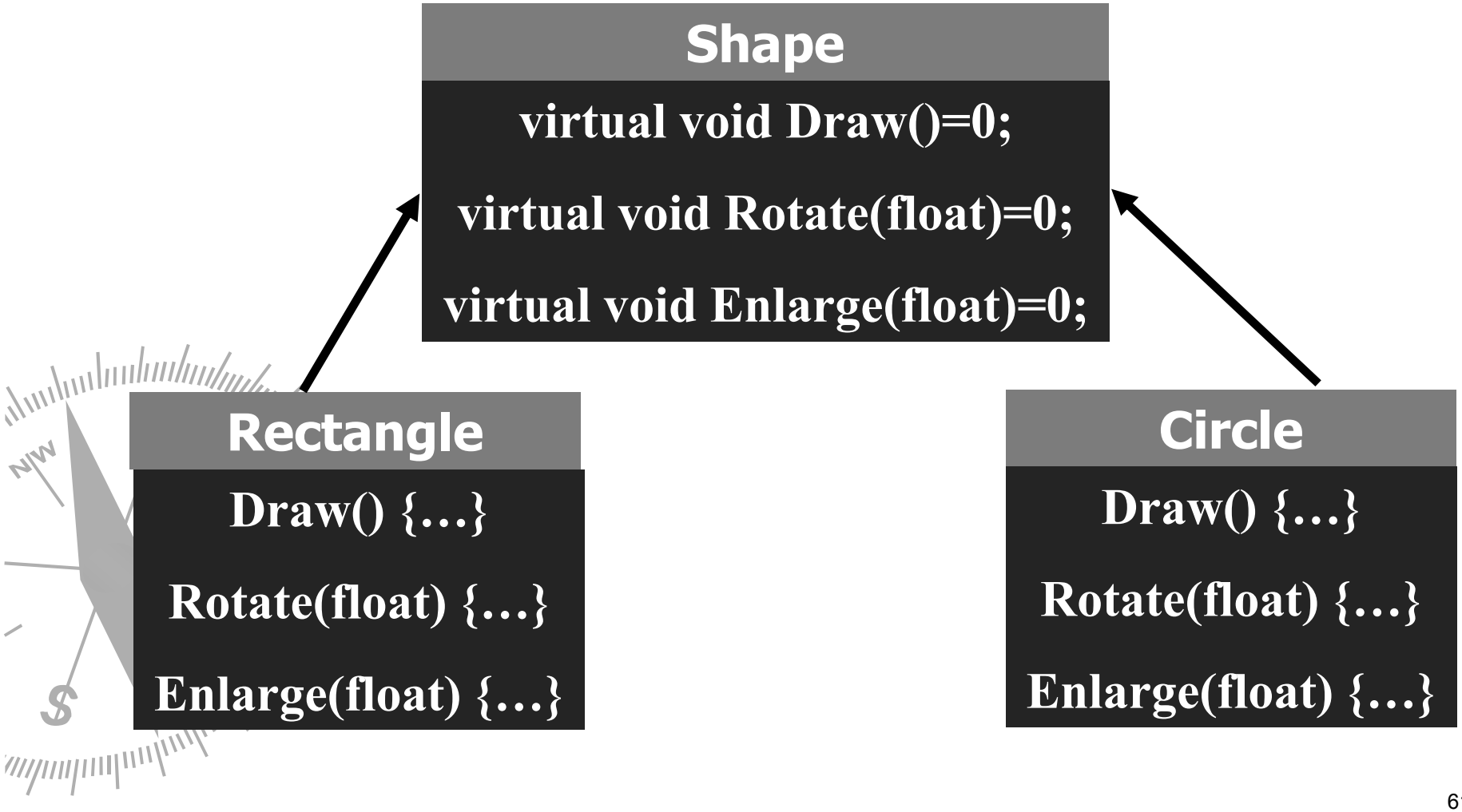
Abstract classes & Pure virtual methods

- אנו רוצים את מחלקת הבסיס רק בשביל לייצג ממשק (interface) משותף לכל צאצאיה!
  - למחלקת Employee יש משמעות וגם למחלקה הנגזרת ממנה Manager. לעומת זאת למחלקת Shape אין שום משמעות, אלא רק לצאצאים שלה!
  - במצב זה (Shape) אנו לא רוצים שמישהו באמת ייצר אובייקט מסוג האב, אלא רק ישתמש במצביעים (או רפרנס) של המחלקה על מנת לעשות upcasting – כדי שיהיה לנו אפשרות לכתוב קוד כללי שמשתמש בכל הצאצאים (דרך הממשק המשותף).
- ברגע שהגדרנו לפחות מתודה *pure virtual* אחת בתוך מחלקה – המחלקה אוטומטית נהפכת למחלקה אבסטרקטית (*abstract class*).
  - ניתן לזהות מתודה pure virtual משום שיהא מוגדרת בווירטואלית ולאחר החתימה שלה מופיע 0=.

Abstract classes & Pure virtual functions cont…

- אי אפשר לייצר אובייקט ממחלקה אבסטרקטית! (שגיאת קומפילציה!).
  - שימו לב שאי אפשר לשלוח אובייקט אבסטרקטי by value! **למה?**
- כאשר יורשים ממחלקה אבסטרקטית חייבים לממש (implement) את כל המתודות הווירטואליות הטהורות, או שגם המחלקה היורשת תהיה מחלקה אבסטרקטית.
- יוצרים מחלקה אבסטרקטית כאשר אנו רוצים ליצור מספר מחלקות באות ממשק זהה ולתפעל את כולם דרך הממשק המשותף, אך למממשק המשותף אין משמעות בפני עצמו או שאין צורך לממש אותו (מימוש מלא).

Abstract classes & Pure virtual functions cont…



Virtual Destructor

- נניח שאנחנו רוצים למחוק אובייקט ממחלקה נגזרת שהוקצה דינמית.
- אנו נעשה זאת על ידי delete למציע שמצביע עליו.
- אם המצביע הוא מסוג מחלקת האב, הקומפיילר, בזמן קומפילציה יכול לקשר אותו רק עם ה-Dtor של מחלקת האב.
- **הבעיה:** אנחנו רוצים לקורא להורס של הבן, כדי שינקה הכל כמו שצריך.
- **נשמע מוכר?** זאת אותה בעיה שבשבילה השתמשנו במתודות ווירטואליות!
- **פתרון:** נגדיר את ההורס כווירטואלי!

האם ניתן לממש פונקציה pure virtual?

```
class A {
public:
    A() {cout << "A's CTOR\n";}
    ~A() {cout << "A's DTOR\n";}
    virtual void f() =0;
};

void A::f()  {cout << "A's f()\n";}

class B : public A {
public:
    B() {}
    ~B() {}
    void f() {cout <<"B's f()\n";}
};
```

```
int main() {
    B b;
    b.f();

    b.A::f();
      //...
}
```

```
//output:
A's CTOR
B's f()
A's f()
A's DTOR
```

בעיה:

- נסתכל על המחלקה הבאה:

```
class Pet {
public:
        Pet() { //... }
        ~Pet() { //... }
        virtual void eat() {//default eat}
        virtual void sleep() {//default sleep}
        virtual void clean() {//default clean}
        virtual void makeSound() {//default makeSound}
        //...
};
```

- נרצה שלא ניתן יהיה ליצור אובייקטים מסוג Pet אלא רק מסוג הילדים שלו.
- ← ניצור פונקציה שתהיה pure virtual.
- איזו פונקציה זו תהיה?

פתרון:

Pet(),
~Pet()
virtual void eat()
virtual void sleep()
virtual void clean()
virtual void makeSound()

- איזו פונקציה זו תהיה?
- נמלה לא צריך להאכיל, דג לא ישן ולא משמיע קול, וחזיר לא מנקים (גם דג ונמלה לא ממש..)
- מה קורה אם אין פונקציה מתאימה להיות pure virtual (לכל פונקציה קיים בן שלא יממש אותה)?
- ← ה-DTOR יהיה pure virtual:

virtual ~**Pet**() = 0;

- בעיה - DTOR חייב מימוש...
- פתרון – מממשים את ה-DTOR למרות שהוא pure virtual

<u>חובה</u> לממש DTOR שהוא pure virtual!