



קבלנות משנה - על ירושה, טענות וחוזים

אוהד ברזילי
תכנות מתקדם בשפת Java
אוניברסיטת תל אביב

ירושה וטענות (assertions)

- תנאי קדם, תנאי בתר ושמורות שהוגדרו עבור מחלקה תקפים גם לגבי צאצאיה, ועשויים להשתנות
- עצם ממחלקה נגזרת המוצבע ע"י עצם (מצביע או הפנייה) מטיפוס מחלקת הבסיס צריך לקיים את שמורת מחלקה הבסיס
- מכאן ששמורה של כל מחלקה יכולה להיות שווה או חזקה יותר משמורת הוריה



קבלנות משנה

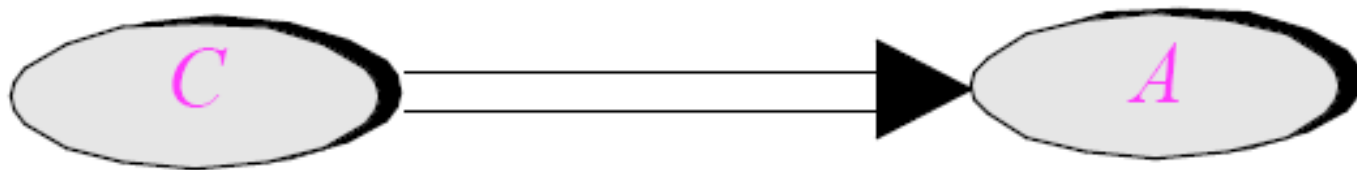
□ מחלקת C היא לקוחה של מחלקה A, כלומר:

■ יש ל-C הפנייה ל-A (אחד השדות)

או

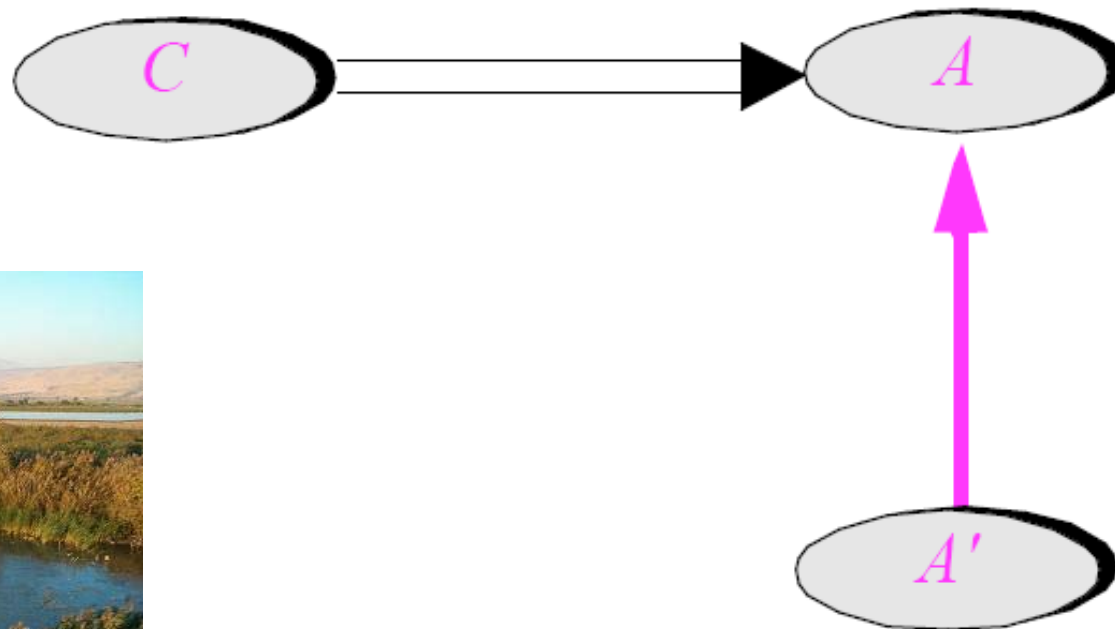
■ אחת המתודות של C מקבלת פרמטר מטיפוס A (הפנייה ל A)

□ C מכירה את השמורה של A ומצפה מ A לקיים אותה



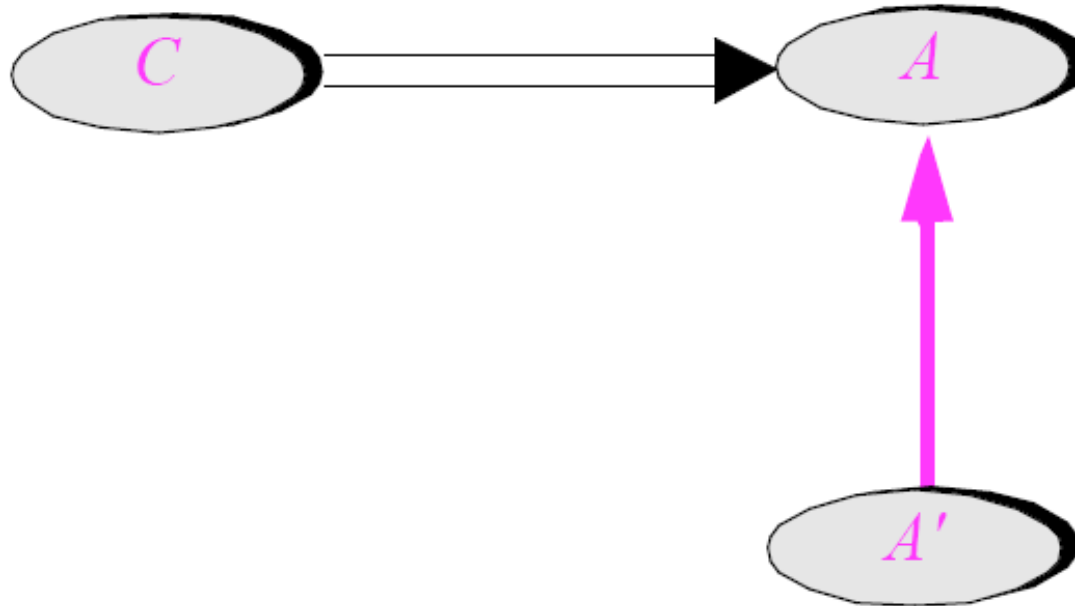
קבלנות משנה - השמורה

- בפועל, המצביע ל- A מצביע ל- A' , מחלקה הנורשת מ- A
- ברור שכדי לקיים יחסים פולימורפים תקינים על A' לקיים לפחות את שמורת A



קבלנות משנה – תנאי קדם ובתר

- המחלקה A' מסתירה (overrides) רוטינה של A
- מה יש לדרוש מתנאי הקדם והבתר של המתודה החדשה ביחס לאלו של הרוטינה המקורית?



r is
require
 α
...
ensure
 β
end

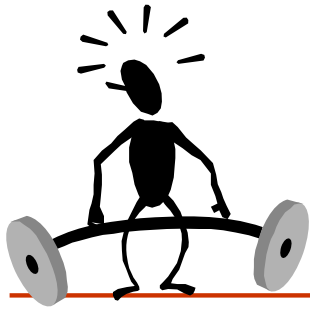
r^{++} is
require
 γ
...
ensure
 δ
end

דוגמא

□ בתוך המחלקה Client מופיע הקוד הבא:

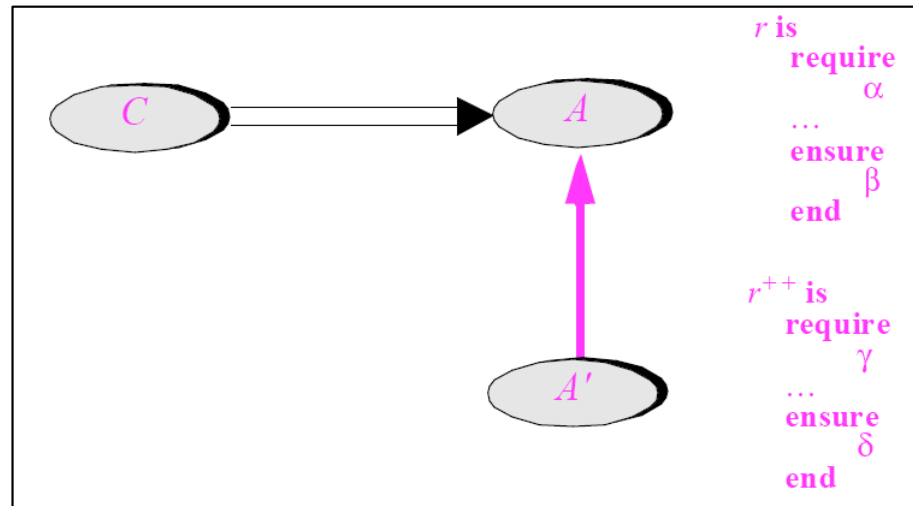
```
public class Client {  
    ...  
    public static void g(String args[])  
    {  
        List<String> l = Arrays.asList(args);  
        ...  
    }  
}
```

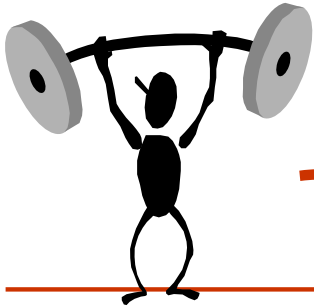
- בדוגמא זו Client הוא הלקוח (C) ו- List הוא הספק (A)
- ואולם ברור ש - l מצביע בפועל לעצם ממחלקה שממשת את List (אולי ArrayList). מחלקה זו היא קבלנית משנה (A')
- הלקוח, שאינו מכיר את קבלן המשנה שלו, מצפה ממנו לעמוד בחוזה המקורי (החוזה מול הספק)



קבלנות משנה – תנאי קדם

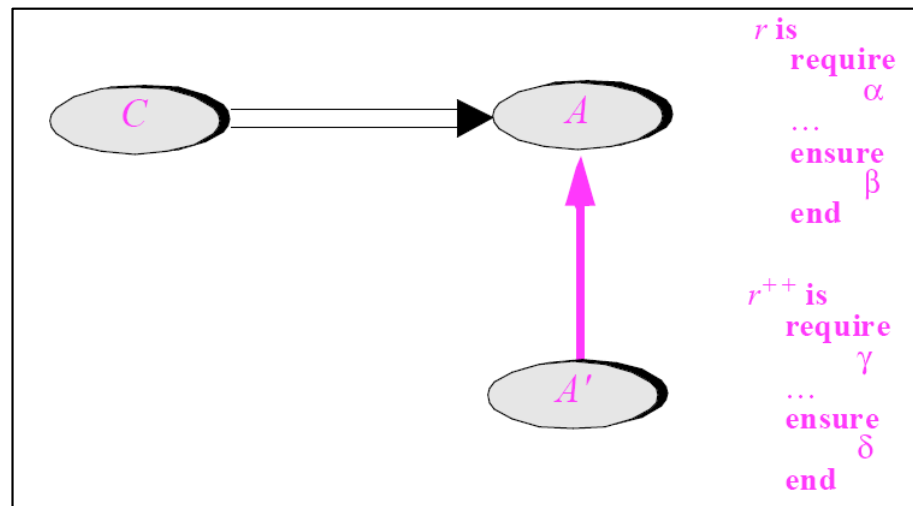
- נתבונן בקריאה $l.r()$ המופיעה במחלקה C
- על C לקיים את תנאי הקדם של $A.r()$, היא כלל אינה מכירה את המחלקה A' ואינה יודעת על קיום $A'.r()$
- לכן על תנאי הקדם המוגדר במחלקה הנגזרת להיות שווה או חלש יותר מתנאי הקדם המקורי





קבלנות משנה – תנאי בתר

- משיקולים דומים על תנאי הבתר של המחלקה הנגזרת להיות שווה או חזק יותר מתנאי הבתר המקורי
- ללקוח C 'הובטח' β ע"י A ואסור שמאחורי הקלעים יסופק δ החלש ממנו
- מנגנון זה מכונה "קבלנות משנה" (subcontracting)



השמורה האפקטיבית

- השמורה ה'אמיתית' של מחלקה מורכבת מ AND לוגי של כל הטענות המופיעות בשמורת אותה מחלקה ובכל הוריה לאורך עץ הירושה
- אם עבור רמה (מחלקה) מסוימת בעץ הירושה לא הוגדרה שמורה, ניתן להתייחס לשמורה שלה כ- TRUE
- כותב מחלקה יכול להגדיר את השמורה שלה בצורה מרומזת (implicit) ע"י ציון הטענות החדשות בלבד

תנאי קדם אפקטיבי

- תנאי הקדם ה'אמיתי' של מתודה שהוגדרה מחדש במחלקה כלשהי, הוא ה OR הלוגי של כל תנאי הקדם של מתודה זו בכל הוריה של אותה מחלקה לאורך עץ הירושה
- אם עבור רמה (מחלקה) מסוימת בעץ הירושה לא הוגדר תנאי קדם למתודה זו, ניתן להתייחס לתנאי הקדם שם כ- FALSE
- כותב תנאי הקדם של המתודה שהוגדרה מחדש במחלקה כלשהי, יכול להגדיר אותו בצורה מרומזת (implicit) ע"י ציון הטענות החדשות בלבד

תנאי בתר אפקטיבי

- תנאי הבתר ה'אמיתי' של מתודה שהוגדרה מחדש במחלקה כלשהי הוא ה AND הלוגי של כל תנאי הבתר של מתודה זו בכל הוריה של אותה מחלקה לאורך עץ הירושה
- אם עבור רמה (מחלקה) מסוימת בעץ הירושה לא הוגדר תנאי קדם למתודה זו, ניתן להתייחס לתנאי הקדם שם כ- TRUE
- כותב תנאי הבתר של המתודה שהוגדרה מחדש במחלקה כלשהי יכול להגדיר אותו בצורה מרומזת (implicit) ע"י ציון הטענות החדשות בלבד

דוגמא

```
public class MATRIX {  
    ...  
    /** inverse of current with precision epsilon  
     * @pre epsilon >= 10 ^(-6)  
     * @post (this.mult($prev(this)) - ONE).norm <= epsilon  
     */  
    void invert(double epsilon);  
    ...  
}
```



דוגמא



```
public class ACCURATE_MATRIX extends MATRIX {  
    ...  
    /** inverse of current with precision epsilon  
     * @pre epsilon >= 10(-20)  
     * @post (this.mult($prev(this)) - ONE).norm <= epsilon/2  
     */  
    void invert(double epsilon);  
    ...  
}
```

□ בשפת Eiffel כדי להדגיש שהחוזה של מתודה שהוגדרה מחדש אינו עומד בפני עצמו אלא תלוי בהיררכיה החליפו את התגיות require ו-ensure ב-require else ו-ensure then בהתאמה

ירושה וחריגים

- אותו היגיון של יחסי ספק, לקוח וקבלן משנה מתקיים גם בנוגע לטיפול בחריגים
- קבלן משנה (מחלקה יורשת, הדורסת מתודה) אינו יכול לזרוק מאחורי הקלעים חריג שלא הוגדר במתודה הנדרסת
- למתודה הדורסת מותר להקל על הלקוח ולזרוק פחות חריגים מהמתודה במחלקת הבסיס שלה

שאלה מתוך מבחן

```
public class A {  
    public float foo(float a, float b) throws IOException{  
    }  
}
```

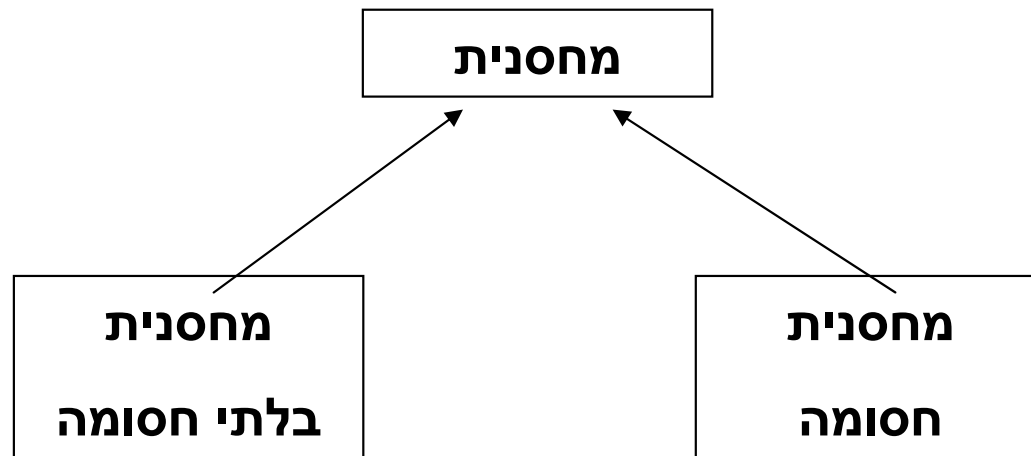
```
public class B extends A {  
    ...  
}
```

Which of the following methods can be defined in B:

- 1. float foo(float a, float b){...}
- 2. public int foo(int a, int b) throws Exception{...}
- 3. public float foo(float a, float b) throws Exception{...}
- 4. public float foo(float p, float q){...}

תנאי קדם מופשט

□ מהי ההיררכיה בין 3 המחלקות: מחסנית, מחסנית חסומה, מחסנית בלתי חסומה?



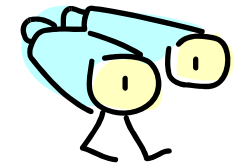
□ מה יהיה תנאי הקדם של המתודה put במחלקה מחסנית?

תנאי קדם מופשט

- תנאי הקדם לא יכול להיות ריק (TRUE) כי אז הוא יחזק ע"י המחסנית החסומה
- תנאי הקדם צריך להיות `!full()` כאשר `full()` היא מתודה המחזירה תמיד `false`, שתוגדר מחדש במחלקה מחסנית חסומה להחזיר `count() == capacity()`
- תנאי קדם המכיל מתודות שנדרסות במורד הירושה נקרא *תנאי קדם מופשט*
- למרות שתנאי הקדם הקונקרטי אכן מתחזק ע"י המחסנית החסומה תנאי הקדם המופשט נשאר ללא שינוי

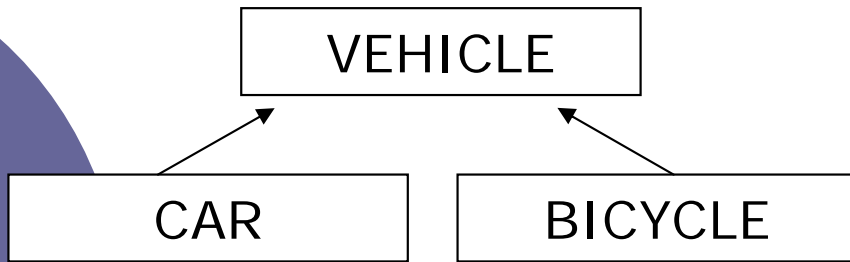
תנאי קדם מופשט

- כאשר מחלקת הבסיס מופשטת, תנאי קדם טריויאליים מחייבים לפעמים *ראייה לעתיד*, כדי שלא יחזקו במחלקות נגזרת
- ראייה לעתיד אינה דבר מופרך במחלקות מופשטות
- נתבונן בדוגמא נוספת: מערכת תוכנה אשר מיוצגים בה כלי תחבורה שונים כגון מכונית, אווירון ואופניים



ראייה לטווח רחוק

- האבולוציה של היררכית מחלקות כלי הרכב לא מתחילה בגזירת מחלקות קונקרטיות שיירשו מ VEHICLE
- הגיוני יותר שבמהלך מימוש ו\או עיצוב המחלקות CAR ו- AIRPLANE נגלה שיש להן הרבה מן המשותף, וכדי למנוע שכפול קוד ניצור מחלקה שלישית - VEHICLE שתכיל את החיתוך של שתיהן
- אף כלי רכב אינו רק VEHICLE
- בראייה זו, לא מוגזם לדרוש ממחלקה מופשטת ניסוח תנאי קדם מופשט



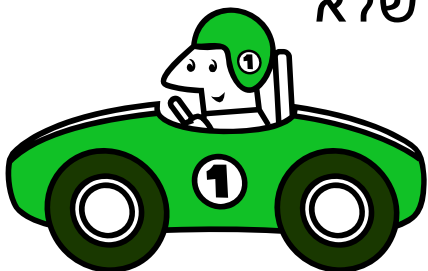
○ מהו תנאי הקדם של המתודה `go()` של המחלקה `VEHICLE` ?

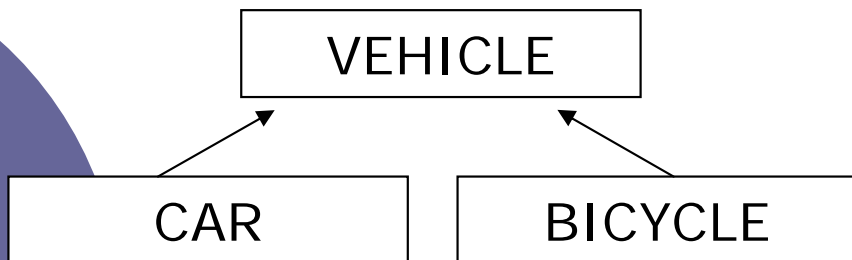
○ על פניו – אין כל תנאי קדם לפעולה מופשטת

○ מה עם המחלקה `CAR` ? – לה בטח יש דרישות כגון `hasFuel()`

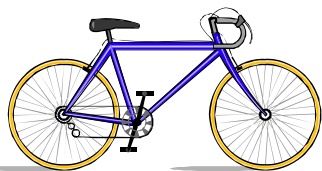
○ מה עם המחלקה `BICYCLE` ? – לה בטח יש דרישות כגון `hasAir()`

○ איך `VEHICLE` תגדיר תנאי קדם ל `go()` גם כללי מספיק וגם שלא יחוזק ע"י אף אחד מירשותיה?





- מתודה בולאנית כגון `canGo()` תעשה את העבודה
- המתודה תוגדר כמחזירה `TRUE` עבור `VEHICLE` (או שתוגדר כ `abstract`), ועבור כל אחת מיורשותיה תוגדר לפי המחלקה האמורה
- בעצם המתודה `go()` היתה צריכה להיקרא `"go_because_you_can()"` וכך לא היתה כל הפתעה בתנאי הקדם "המוזר"



מתי לרשת? - דיון

□ המחלקה CAR_OWNER עשויה לרשת מ PERSON אבל עדיף שתהיה לקוחה של CAR

□ יחס is-a לעומת יחס is-part-of

□ פרט למנגנון הרב-צורתיות (polymorphism) ירושה לעולם אינה הכרחית

מתי לרשת? - דיון

□ במקום ש B יירש מ-A , ל- B יכולה להיות התכונה A (שדה מוכל או מצביע)

□ To be is also to have אבל לא להיפך (משאית היא מכונית כלומר חלק בה הוא מכונית)

□ לפעמים נוח לשאול "האם יכולים להיות לו שניים?"
■ לדוגמא: למכונית יש מנוע

□ ירושה או מופע?