ירושה

ירושה היא תכונה של תכנות מונחה עצמים המאפשרת להרחיב את הגדרת המחלקה ולהשתמש במחלקות בצורה כללית יותר.

1. <u>ירושה</u>

ירושה היא תכונה שאומרת שמחלקה אחת היא סוג של מחלקה אחרת עם תוספות. בירושה אנחנו משתמשים במחלקה מסויימת כבסיס למחלקות אחרות. כלומר, כל התכונות והשיטות של מחלקת הבסיס קיימות גם במחלקה היורשת, שיכולה להוסיף גם תכונות ושיטות משלה.

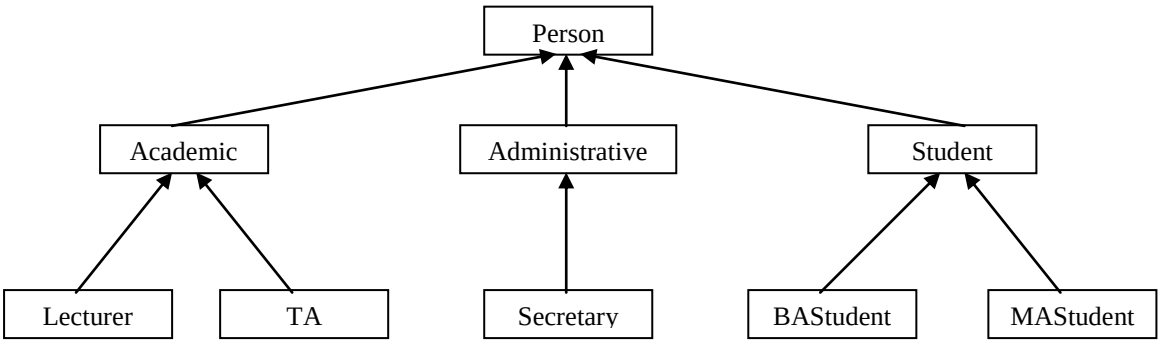
למשל, נניח שבאוניברסיטה מסויימת רוצים לשמור מידע על כל האנשים שנמצאים באוניברסיטה. נוכל להגדיר מחלקה אחת שנקראת `Person`. מחלקה זו תייצג בן אדם. לבן אדם יכולות להיות כל מיני תכונות, למשל שם, מספר ת.ז., תאריך לידה וכו'. כמו כן יכולות להיות לו מספר שיטות, בין היתר שיטות האיחזור והקביעה לתכונות (`getName, setName` וכו').

כעת נוכל להגדיר מחלקה נוספת שתיקרא `Academic`. מחלקה זו תשמור נתונים על עובדי האוניברסיטה. ברור שכל עובד אוניברסיטה הוא בן אדם, ולכן אמורות להיות לו התכונות של `Person`, אולם לעובדי האוניברסיטה יש גם תכונות ושיטות אחרות שלא קיימות לכל בני האדם. למשל, תאריך התחלת עבודה באוניברסיטה, תפקיד, משכורת ועוד. כיוון שאובייקטים מסוג `Academic` הם בעצם `Person` מסוג ספציפי יותר, נוכל לאמר שהמחלקה `Academic` יורשת מהמחלקה `Person`.

2. <u>עץ ירושה</u>

נוכל לתאר את היחסים בין המחלקות השונות ע"י שרטוט שנקרא עץ ירושה. העץ מתאר ברמות העליונות את המחלקות המורישות (מחלקות הבסיס) וברמות התחתונות את המחלקות היורשות.

למשל, כך יראה עץ הירושה עבור מערכת האוניברסיטה:



לפי העץ הנתון אנו רואים את היחסים בין המחלקות: מחלקת האב היא `Person` וממנה יורשות שלוש מחלקות – `Academic` שמייצגת את הסגל האקדמי של האוניברסיטה,

Administrative שמייצגת את הסגל האדמיניסטרטיבי של האוניברסיטה ו-Student שמייצגת את הסטודנטים.

יחס הירושה מאופיין ע"י הביטוי is-a. כלומר, אם נוכל לאמר שמחלקה אחת היא סוג של מחלקה אחרת, אז נדע שישנו יחס ירושה בין המחלקות. למשל, המשפט Student is-a Person הוא נכון כי כל סטודנט הוא בן אדם, ולכן ישנה ירושה בין המחלקות. שימו לב שההיפך לא בהכרח נכון – Person is-a Student, כיוון שלא כל בן אדם הוא סטודנט, ולכן הירושה לא מסומנת בכיוון הזה.

הירושה לא מוגבלת לזוג אחד של מחלקות, ואפשר להמשיך ולהוריש הלאה. למשל, בדוגמא שלנו אנו רואים שמרצה הוא סוג של סגל אקדמי שהוא סוג של בן אדם, ולכן כל מרצה מקבל את כל ההגדרות של כל המחלקות שמעליו, עד למחלקת הבסיס.

אנו קוראים למחלקה המורישה מחלקת בסיס (Base Class) או מחלקת אב, ולמחלקה היורשת תת מחלקה (Sub Class) או מחלקות בנים. שימו לב שכל תת מחלקה יכולה להפוך בתורה למחלקת בסיס עבור מחלקה אחרת.

<u>שאלה 1</u>: אילו דברים יהיו במחלקות הבסיס ואילו דברים במחלקות היורשות?

הרעיון של ירושה הוא פשוט – ככל שאנו עולים במעלה עץ הירושה, אנו נהיים יותר ויותר כלליים. מה שנמצא במחלקת הבסיס חייב להיות משותף לכל המחלקות ותתי המחלקות שיורשות ממנה, ולכן הדברים שאנו יכולים לאמר על מחלקת הבסיס חייבים להיות כלליים. למשל, מה שנמצא במחלקת Person חייב להיות משותף לכל שאר המחלקות בעץ (למשל, גם לסטודנטים מתואר ראשון וגם למתרגלים) ולכן אנו נכתוב במחלקה רק את הדברים שהם באמת כלליים מספיק.

מצד שני, ככל שאנו יורדים מטה בעץ אנו נהיים יותר ויותר ספציפיים. למשל, הדברים שאנו יכולים להגיד על סגל אקדמי הם יותר ספציפיים ממה שניתן להגיד על Person. נכון שכל עובד בסגל האקדמי הוא סוג של Person, אבל יש לו גם דברים נוספים (כמו למשל, משכורת) שאין לכל האובייקטים מסוג Person (למשל, לסטודנטים אין את התכונה הזאת). ואם יורדים עוד רמה למטה, אז בוודאי שמה שנוכל להגיד על אובייקטים מסוג Lecturer הוא הרבה יותר ספציפי ממה שיכלנו להגיד על אובייקטים מסוג Academic ועל אובייקטים מסוג Person.

3. <u>ירושה מרובה</u>

נניח שהיינו רוצים להוסיף מחלקה שמייצגת "מרכז קורס". מרכז קורס חייב להיות איש אקדמיה בעצמו, אבל בנוסף, יש לו גם אחריות אדמיניסטרטיבית (למשל, הוא שוכר מרצים לקורס, מעביר חוזים וכו'). לפי הגדרת הירושה נוכל לאמר ש Coordinator is-a Academic וגם ש- Coordinator is-a Administrative. כלומר מרכז הקורס אמור לרשת משתי מחלקות שונות ולקבל משתיהן את התכונות שלהן.

מצב כזה נקרא ירושה מרובה, ובג'אווה המצב הזה אינו אפשרי. בג'אווה כל מחלקה יכולה לרשת רק ממחלקה אחת! (ההיפך כמובן לא נכון – כל מחלקה יכולה להוריש את עצמה למספר לא מוגבל של מחלקות בנים).

נראה בהמשך איך אפשר להתמודד (במידה מסויימת) עם מצבים שדורשים ירושה מרובה, אבל לעת עתה נזכור שלכל מחלקה יש רק אבא אחד.

4. Object

מה משותף לכל בני האדם, הסטודנטים, החיות, הבתים, המלבנים וכו'? בסופו של דבר, כולם אובייקטים. למעשה המשפט הבא יהיה נכון **תמיד**: Something is-a Object, עבור כל מחלקה שנבחר, כי כל מחלקה היא בעצם סוג של אובייקט. ולפי הגדרת הירושה אנו יודעים שבמצב כזה כנראה שיש יחס ירושה בין המחלקה לבין Object.

בג'אווה אכן קיים יחס כזה, ומוגדרת מחלקה בשם Object שמתפקדת כמחלקת האב של כל המחלקות באשר הן. כלומר, לכל מחלקה בג'אווה יש אב קדמון שהוא Object והוא מתחיל את עץ הירושה.

שאלה 2: אילו דברים יהיו במחלקה Object? (זכרו שככל שעולים במעלה העץ צריכים להיות כלליים מאוד, ו-Object נמצא בראש העץ).

5. הגדרת ירושה בג'אווה

נוכל לציין שמחלקה מסויימת יורשת ממחלקה אחרת ע"י שימוש במילה השמורה extends, כך:

```
public class Person
{…}
public class Academic extends Person
{…}
```

שאלה 3: נניח שהיתה מחלקה נוספת בשם Human. האם היינו יכולים לשכתב את הגדרת המחלקה Academic כך:

```
public class Academic extends Person, Human
```

שאלה 4: ראינו קודם שהמחלקה Object היא הבסיס לכל עץ ירושה. למה לא ציינו את זה אף פעם? למשל, למה לא כתבנו:

```
public class Person extends Object
```

6. תחום ההכרה של תכונות בירושה

ברגע שמחלקה מסויימת יורשת ממחלקה אחרת, היא מקבלת את כל התכונות והשיטות של מחלקת האב (אין צורך לרשום אותן, הן כבר שם). למשל:

```
public class Person
{
   private String _name;
}
public class Academic extends Person
{
   private double _salary;
}
```

למחלקה Academic יש שתי תכונות – התכונה _salary שמוגדרת בתוכה, וגם התכונה _name שמגיעה בירושה מ-Person.

<u>שאלה 5</u>: האם בתוך המחלקה Academic ניתן להשתמש בתכונה _name?

שימו לב – תכונה שמוגדרת כ-private מוכרת רק בתחומי המחלקה בה היא מוגדרת! כלומר, אפילו שהמחלקה Academic יורשת מ-Person, עדיין אין לה גישה לתכונה _name כיוון שהיא מוגדרת כפרטית. זהו מצב בעייתי, כיוון שלפי הגדרת הירושה, הרי Academic היא סוג של Person ולכן הגיוני שתהיה לה גישה ישירה לתכונה _name (שהיא גם תכונה שלה).

כדי לפתור את המצב מוגדר בג'אווה מאפיין גישה נוסף שנקרא protected. תכונה שמוגדרת ב-protected נגישה בתוך תחומי המחלקה שלה וגם בתוך תחומי כל המחלקות שיורשות מהמחלקה! למשל, אם נגדיר את תכונת ה-_name כ-protected היא תהיה נגישה גם ל-Academic וגם ל-Lecturer ול-TA. בעצם התכונה נגישה החל מהמקום שבו היא מוגדרת ומטה בעץ הירושה.

שימו לב שתכונה שמוגדרת כ-protected מוכרת גם לכל המחלקות שנמצאות באותו package, אבל אנו לא מתעסקים בזה בקורס הזה.

7. <u>הפעלת הבנאים בירושה</u>

אנו יודעים שבג'אווה, לכל מחלקה יש בנאי (לפחות אחד). כאשר ישנה ירושה, נשאלת השאלה אילו מהבנאים מופעל ומתי.

באופן עקרוני, הכלל בירושה הוא ברור – בנאי של תת מחלקה **חייב** לקרוא קודם לבנאי של מחלקת האב ורק אח"כ הוא משלים את פעולתו. למשל, נניח שנתון הקוד הבא:

```
public class Person
{
   protected String _name;
   public Person()
   {
        _name = "";
        System.our.println("In Person");
   }
}
public class Academic extends Person
{
   protected double _salary;
```

```
    public Academic()
    {
        _salary = 0.0;
        System.out.println("In Academic");
    }
}
```

ונתון ה-main הבא:

```
public class Tester {
    public static void main()
    {
        Academic a = new Academic();
    }
}
```

הפלט שנקבל יהיה:

```
In Person
In Academic
```

התהליך שהתרחש: בתוך ה-main היתה פניה לבנאי הריק של Academic. כשהבנאי נקרא, הדבר הראשון שהוא עושה הוא לפנות לבנאי הריק של Person. הבנאי של Person קודם כל פונה לבנאי הריק של Object. כעת כשהגענו לראש עץ הירושה הבנאים מתחילים להתבצע בסדר הפוך – הבנאי הריק של Object מסיים את עבודתו, ואז חוזרת השליטה לבנאי של Person. הוא מבצע את האיתחול ומדפיס את השורה In Person על המסך, ואז חוזרת השליטה לבנאי הריק של Academic שמבצע גם הוא את פקודותיו ומסיים.

<u>שאלה 6</u>: בהינתן הקוד הבא, מה יהיה סדר הפעלת הבנאים ביצירת אובייקט מסוג Lecturer?

```
public class Lecturer extends Academic {
    public Lecturer()
    {
        …
    }
}
```

<u>שאלה 7</u>: נניח שבמחלקה Person לא היה מוגדר בנאי ריק, אבל היה מוגדר בנאי אחר:

```
public Person(String name)

{ _name = name; }
```

מה היה עכשיו סדר הפעלת הבנאים ביצירת אובייקטים מסוג Academic?

8. <u>הפעלה יזומה של בנאים</u>

אנו יודעים שלמחלקות יכולים להיות סוגים רבים של בנאים, שאותם אפשר להפעיל לפי בחירה. מצד שני, ראינו קודם שביצירת אובייקטים בירושה, ג'אווה מפעילה בברירת מחדל רק את הבנאי הריק. למשל, נניח שלמחלקה Person ישנם שני בנאים, הבנאי הריק, ובנאי שמקבל מחרוזת של שם (כפי שראינו קודם).

נרצה להוסיף למחלקה Academic בנאי שחתימתו:

```
public Academic(String name, double salary)
```

שאלה 8: נוכל עדיין לממש את אתחול השם בתוך הבנאי של Academic כך:

```
_name = name;
```

כיוון ש-_name מוגדר כ-protected ול-Academic יש גישה אליו. מה בכל זאת הבעייתיות בצורה כזאת של אתחול משתני מחלקת הבסיס?

כיוון שאנו יודעים שהבנאי של האב חייב להיקרא קודם, נוכל להפנות את הקריאה לבנאי אחר מהבנאי הריק ע"י שימוש במילה השמורה super. מילה זו מציינת את המצביע לאבא הישיר של המחלקה, וכדי להפעיל את הבנאי של האב, נוכל להעביר את הפרמטרים הרצויים, כך:

```
public Academic(String name, double salary)
{
   super(name);
   _salary = salary;
}
```

הפקודה super בעצם מכילה הפניה מפורשת לבנאי כלשהו (לפי הפרמטרים הקומפיילר יודע לאיזה בנאי לפנות). שימו לב שהקריאה ל-super אם קיימת, חייבת לבוא בשורה הראשונה של הבנאי של הבן, אחרת תהיה שגיאת קומפילציה!

שאלה 9: למה לא ניתן לכתוב את הקריאה ל-super לא בשורה הראשונה של בנאי הבן?

שאלה 10: האם הקוד הבא יעבור קומפילציה?

```
public Academic(String name, double salary)
{
   super(salary);
}
```

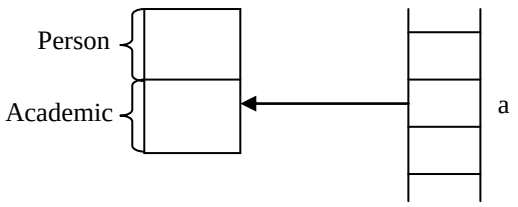
שאלה 11: מה משמעות הביטוי super(); ?

9. מבנה הזיכרון בירושה

קל יותר להבין את סדר הפעלת הבנאים והשיטות (בהמשך) אם נדמיין את מבנה הזיכרון של אובייקטים שנוצרו בירושה. למשל, אם ב-main היתה שורת הקוד הבאה:

```
Academic a = new Academic();
```

מבנה הזיכרון יראה כך:



כלומר, בלוק הזיכרון שנוצר כדי להכיל אובייקט מסוג Academic מכיל בעצם שני חלקים – אחד ל-Person ואחד ל-Academic (ברצף!), ותא הזיכרון a מצביע מהמחסנית על החלק של Academic.

נשתמש בייצוג זה בהמשך.

10. דריסה (overriding) של שיטות

מחלקה יורשת יכולה להגדיר מחדש שיטות שכבר הוגדרו במחלקת האב. למשל, נניח שבמחלקה Person קיימת השיטה:

```
public void goToWork()
{
   System.out.println("I'm going to work");
}
```

כעת נוכל להוסיף במחלקה Academic שיטה עם חתימה זהה:

```
public class Academic extends Person
{
   public void goToWork()
   {
        System.out.println("I'm going to teach");
   }
}
```

צורת הגדרה זו נקראת **דריסה** (overriding) של שיטות. אנו אומרים שהמחלקה Academic דורסת את השיטה goToWork של המחלקה Person ובעצם מגדירה אותה מחדש. כעת, כשניצור אובייקט מסוג Academic איזו מהשיטות יופעלו?

```
Academic a = new Academic();
a.goToWork();
```

כיוון שהאובייקט הוא מסוג Academic אז השיטה שתופעל היא השיטה שמוגדרת במחלקה זו. שימו לב לכלל – ברגע שנכנסים לזיכרון האובייקט, מופעלת השיטה התחתונה ביותר ("כוח המשיכה"), או במילים אחרות, השיטה העדכנית ביותר.

שאלה 11: האם נוכל בכל זאת מתוך ה-main ליצור אובייקט מסוג Academic אבל להפעיל את השיטה goToWork של Person?

התשובה היא לא! ברגע שמחלקה דרסה שיטה, אזי היא מגדירה התנהגות יותר ספציפית לאותה שיטה, ולכן השיטה הנדרסת כבר לא רלבנטית. שימו לב שמתוך המחלקה הדורסת יש אפשרות להפעיל את השיטה של האב, באמצעות שימוש בפקודה super, למשל:

```
public void goToWork()
{
     super.goToWork();
     System.out.println("I'm going to teach");
}
```

גם פה, כמו בהפעלת בנאים, המילה super משמשת להפניה ישירה למחלקה המורישה, אבל פה, בניגוד להפעלת בנאים, השימוש במילה נעשה על מנת להפעיל שיטה. לכן גם אין הכרח שפקודה זו תופיע בשורה הראשונה.

שאלה 12: מה ההבדל בין super לבין this? האם יכול להיות מקרה שאחד מהם הוא null?

שאלה 13: מאיזו סיבה נרצה להגדיר מחדש שיטות שכבר הוגדרו במחלקת האב?

שאלה 14: מה ההבדל בין overriding ל-overloading?

שימו לב שחתימת השיטה הדורסת חייבת להיות בדיוק כמו חתימת השיטה הנדרסת, למעט מאפיין הגישה. שיטה שדורסת יכולה להיות ליברלית יותר מהשיטה הנדרסת, אבל לא פחות. למשל, אם במחלקת האב היתה שיטה שהוגדרה כ-private אזי במחלקה היורשת אפשר לדרוס את השיטה ולהגדירה כ-protected או כ-public. לעומת זאת, אם השיטה במחלקת האב הוגדרה כ-public אין שום אפשרות במחלקת הבן לשנות את מאפיין הגישה הזה.

11. מחלקות ושיטות מופשטות

ניתן להגדיר מחלקה כמופשטת (abstract) ע"י שימוש במילה השמורה abstract בראש המחלקה:

```
public abstract class Person {…}
```

המשמעות של מחלקה מופשטת היא שלא ניתן ליצור ממחלקה זו אובייקטים. למשל, הקוד הבא ב-main לא יעבור קומפילציה כיוון ש-Person מוגדרת כמופשטת:

```
Person p = new Person(); // doesn't compiled
```

מצד שני, ניתן לרשת ממחלקה מופשטת.

שאלה 15: מאיזו סיבה נרצה להגדיר מחלקה שאי אפשר ליצור ממנה אובייקטים?

מחלקה מופשטת יכולה להכיל גם שיטות מופשטות. שיטה מופשטת היא שיטה שאין לה גוף (החתימה שלה מסתיימת בנקודה פסיק), אולם כל מחלקה שיורשת מהמחלקה המופשטת **חייבת** לממש את כל השיטות המופשטות שלה! ניתן להגדיר שיטה מופשטת ע"י הוספת המילה abstract בחתימת השיטה:

```
public abstract class Person
{
   public abstract void goToWork();
}
public class Academic extends Person
{
   public void goToWork()
   {…}
}
```

כיוון שהמחלקה Academic ירשה מהמחלקה המופשטת Person, היא חייבת לדרוס את השיטה goToWork ולספק לה מימוש, אחרת הקוד שלה לא יעבור קומפילציה.

שאלה 16: מה היתרון להגדיר שיטה בלי גוף שמחוייבת להתממש במחלקות היורשות?

שימו לב שברגע שמחלקה מכילה לפחות שיטה אחת מופשטת, היא הופכת בעצמה להיות מופשטת.

הדרך היחידה של מחלקה להתחמק ממימוש השיטות האבסטרקטיות היא להפוך בעצמה לאבסטרקטית. כך המחלקה פטורה ממימוש השיטות, והיא יכולה לממש או לא לממש את השיטות כרצונה. למשל –

```
public abstract class Academic extends Person
{
// No need to override goToWork()
}
```

שאלה 17: מתי נשתמש במנגנון של מחלקה מופשטת שיורשת ממחלקה מופשטת? למה בכלל להגדיר את המחלקות אם הן מופשטות ולא ניתן ליצור מהן אובייקטים?

שאלה 18: האם מחלקה מופשטת יכולה להכיל שיטות לא מופשטות? האם מחלקה מופשטת מכילה בנאי?