**תרגול OOP - 15.11**

**ירושה:**

* נתונה פונקציה במחלקת האב:
  + Overload – פונקציה חדשה עם אותו שם אך עם חתימה קצת אחרת שמוסיפה פונקציונליות לפונקציה המקורית
  + Override – דריסה של הפונקציה
* מתודה וירטואלית היא מתודה שאפשר לדרוס, בJAVA הכל וירטואלי ולכן תמיד ניתן לדרוס (אלא אם נכתוב final)
* ברמת התרגום לbytecode למעשה קוראים לאותו הדבר עבור פונקציה וירטואלית F שדרסנו אותה ע"י מחלקת בן, גם אם נקרא לה מאובייקט של מחלקת בן וגם אם נקרא לה מאובייקט של מחלקת האב.
  + איך זה עובד בפועל? בעזרת טבלה וירטואלית – vtable – כל אובייקט מחזיק טבלה של אובייקט שמתאימה בין הפונקציה שמנסים לקרוא לה לבין הפונקציה שבאמת צריך לקרוא לה.
  + לאובייקט של מחלקת הבן נגדיר בטבלה שקריאה לפונקציה F על האובייקט תיקרא לפונקציה שדרסנו מתוך מחלקת הבן
  + מחלקת הבן מעתיקה את הטבלה של מחלקת האב, מוסיפה לה את המתודות החדשות של מחלקת הבן בלבד, ואם יש override היא משנה את הקריאה לפונקציה המתאימה ממחלקת הבן
  + הטבלה היא לכל אובייקט (כלומר לכל אובייקט יש מצביע לטבלה)
    - אם האובייקט לא של מחלקה יורשת אז הטבלה לא תהיה מאוד מעניינת אולם אם כן יש ירושה אנחנו נראה הבדל בין הטבלאות
  + הJVM יחפש בטבלה לאיזו פונקציה לקרוא לפי האובייקט שקרא לפונקציה
* הרצה עם דגל -verbose
  + Constant pool – רשימה של כל מה שהגדרנו בתוכנית שלנו ומיפוי שלו למספרים
  + תרגום לאסמבלי
  + Line number table – ממפה את מספרי השורות שלנו בקוד לשורות בתרגום לbytecode

**חבילות:**

* חבילות הן מבנה מבוסס תיקיות שמאפשר לאחד קבצי מקור ביחד
  + אפשר לחשוב כמו על היררכיית תיקיות במחשב
* פרויקטים בג'אווה יכולים להיות עצומים וחבילות מאפשרות ליצור סדר, לדעת מה קשור למה וכדומה.
* יתרונות של חלוקה לחבילות:
  + עוזר למתכנת להבין מה קשור למה
  + עוזר להבין איפה אפשר למצוא קבצים, להתמצא בתוך המרחב הענק של הקבצים שיכולים להיות
  + עוזר למנוע קונפליקטים – למשל אם בשני קבצים יש מחלקה עם אותו שם אז בחלוקה לתיקיות זה מתאפשר (אם הם יהיו באותה חבילה זה לא יעבוד)
  + מאפשר לנהל גישה – למה אפשר לגשת מחוץ לחבילה, למה אפשר לגשת בין חבילות
* יצירת חבילה –
  + בחירת שם אינדיקטיבי וברור
  + נוסיף את הפקודה package <name> בשורה הראשונה של כל קובץ שנרצה שיהיה בחבילה
    - אי אפשר לשייך קובץ לכמה חבילות ולכן השורה תופיע פעם אחת בלבד בשורה הראשונה בקובץ
    - כל מה שמוגדר בתוך הקובץ יהיה מחויב לחבילה אליה שויך הקובץ.
* אם לא נכתוב את המילה package בראש הקובץ, הקובץ יהיה למעשה שייך לחבילה אנונימית
  + חבילה אנונימית מתאימה לפרויקטים ממש קטנים או לפרויקטים זמניים – בכל מצב אחר זה לא מה שאנחנו אמורים לעשות
* כל הטיפוסים שמוגדרים באותה חבילה שמוגרים כפומביים ניתן לגשת אליו מחוץ לחבילה בשלוש דרכים:
  + גישה עם השם המלא – שם החבילה, נקודה, שם הקובץ / המחלקה
  + ניתן לעשות import לקובץ מסוים מהחבילה ואז לא נצטרך לרשום את שם החבילה לפני גישה לאיבר בקובץ הזה
  + לייבא pack.\* - את כל החבילה
* אי אפשר לגשת לאיברי חבילות שאינם פומביים
  + אם לא כתבנו modifier לאיבר מסוים האיבר יהיה package private
* חבילות יכולות להיות היררכיות אך זה יהיה רק מבחינה לוגית – הם לא באמת יהיו מוכלים אחד בשני – כלומר Java.A.B – B לא נמצא בתוך A אלא רק משויך אליו לוגית, נהיה חייבים לייבא את Java.A כדי לגשת למה שיש בו מתוך B, ואם נייבא את Java.A לא נייבא את B
* אם אנחנו מייבאים שתי חבילות ששתיהן מכילות מחלקה באותו שם, גם אם נייבא את כל הקבצים והמחלקות בחבילות בעזרת \* עדיין נצטרך לגשת אליהן עם שם החבילה בהתחלה כדי להבדיל ביניהן
* Static import:
  + Import מאפשר לייבא טיפוסים אך לא שדות
  + לעומת זאת אם נוסיף static נוכל לייבא שדה ולא נצטרך לפנות אליו עם שם החבילה והמחלקה
  + ייבוא סטטי צריך להיות בראש הקובץ
  + \*
* נהוג לשים חבילות לפי תיקיות המתאימות לשמות של החבילות – לא חייבים לעשות זאת אך אם לא נעשה נצטרך להגיד לקומפיילר איפה נמצא כל דבר
  + המבנה ההיררכי של תיקיות הקבצים צריך להיות זהה למבנה ההיררכי של החבילות (עד כדדי החלפת נקודה ב\)
* הקמפול שומר על ההיררכיה של קבצי המקור לקבצים שיוצר
* ניתן בקמפול את הדגל -cp, -classpath כדי להגיד לקומפיילר איפה נמצאים כל הדברים שהוא צריך כדי להצליח לקמפל את קבצי המקור שמשתמשים בהם