packages

מבוא

כאשר כותבים תוכנה בהיקף בינוני או גדול, עולה הצורך לחלוקת הקוד לרכיבים תפקודיים. בדרך כלל התוכנה מתחלקת באופן טבעי לרכיבים המבצעים פעולות נפרדות. לדוגמה, יכל להיות לתוכנה רכיב האחראי על פענוח אינפורמציה מקבצים ורכיב נוסף האחראי על חישובים נומריים הדרושים לתוכנה. רכיב תוכנה כזה, מכונה לעיתים מודול (module) . כל מודול מכיל הרבה פרטים הרלוונטיים לו בלבד ואינם חשובים למודולים האחרים של התוכנה. אם כותבים תוכנה גדולה באותו אופן שכתבנו עד כה, נתקלים בבעיה של התנגשות שמות: שם של מחלקה ממודול אחד, זהה לשם ממודול שני. לדוגמה, מקרה בו מוגדרת בשני מודולים המחלקה Parser, באחת במשמעות של מפענח תחבירי של קבצים ובשניה בשביל אינטראקציה עם המשתמש בשפה טבעית. תוכנית לא יכולה לכלול שתי מחלקות בעלות אותו שם אם אין מנגנון שמבדיל ביניהן.

צורך נוסף העולה בחלוקה למודולים קשור להרשאות גישה. המרכיבים השונים של מודול, זקוקים לעיתים להכיר זה את זה בצורה אינטימית. נדרשת הרשאת גישה המאפשרת למרכיבים גישה לשדות של חבריהם למודול אבל עדיין מונעת גישה של קוד ממודולים אחרים.

צורך שלישי הקשור לתוכנה גדולה הוא ארגון קבצי הקוד במערכת הקבצים כך שהתוכנה כולה תהיה נוחה לתחזוקה.

בשביל לענות על שלושת הצרכים שהצגנו (התנגשות שמות, הרשאות וסידור קבצים) כוללת שפת התכנות Java מנגנון של חבילות - packages. כל חבילה , מייצגת מודול בתוכנית.

הצהרת שיוך לחבילה

בכדי להצהיר שהמחלקות בקובץ קוד מסוים, שייכות לחבילה, יש לכתוב את המילה השמורה package ואת שם המחלקה בשורת הקוד הראשונה שאיננה הערה:

file Check.java:

```
// bla bla bla
package mypackage;
class A{..}
class B{..}
class C{..}
```

בדוגמה זו, המחלקות A , B ו C שייכות כולן לחבילה mypackage. מכיוון שאף אחת מהן לא מוצהרת כ - public הן לא תהינה מוכרות מחבילות אחרות. מחלקות שאינן public משמשות רק לצרכים פנמיים של החבילה.

בכל קובץ קוד (ניקרא גם "יחידת קומפילציה" או "יחידת תרגום") יכלה להיות רק מחלקה אחת המוצהרת כ - public, למחלקה זו חייב להיות שם זהה לשם הקובץ (ללא ה-extension). לדוגמה:

file Check.java:

```
// bla bla bla
package mypackage;
class A{..}
public class Check {…}
class B{..}
class C{..}
```

בדוגמה זו יהיה ניתן להתייחס ל Check גם מחבילות אחרות. השם של Check עבור קוד שאינו של המחלקה mypackage הוא mypackage.Check. שם החבילה המופיע לפני שם המחלקה, הוא זה שמונע התנגשות שמות עם מחלקה אחרת בשם Check, השייכת לחבילה אחרת.

כאשר קובץ קוד עובר קומפילציה, נוצר קובץ class. עבור כל מחלקה. קומפילציה של הדוגמה האחרונה, תצור את הקבצים:

A.class B.class C.class Check.class

כידוע, תוכנית ריצה ב - Java היא בעצם אוסף של קבצי class המורצים ע"י המכונה הוירטואלית. במקרה שלפנינו, הקוד של Check.class יוכל להיות מופעל ע"י קבצי class שאינם שייכים לחבילה אך A.class, B.class ו - C.class - לא.

import

כאשר מתייחסים מתוך חבילה אחת למחלקה השייכת למחלקה אחרת, ניתן לעשות זאת ע"י שימוש בשם המלא. לדוגמה:

```
package otherpackage;
class Stam{
        void foo() {
                mypackage.A a = new mypackage.A();
        }
}
```

(הקובץ A.class צריך להיות ממוקם במקום אותו ניתן למצוא כפי שיוסבר בהמשך)
השם המלא אמנם מונע התנגשות שמות אך הוא מסורבל לכתיבה. ניתן להשתמש בפקודה import באופן הבא:

```
package otherpackage;
import mypackage.A;
class Stam{
        void foo() {
                A a = new A();
                //B b; //error: should be mypackage.B
        }
}
```

הפקודה חייבת להופיע לפני כל קוד אחר למעט פקודת ה - package (אם קיימת). לאחר שימוש בפקודה, אפשר להשתמש בשם המקוצר של המחלקה. למחלקות אחרות צריך עדיין להתייחס בשמן המלא. ניתן לבצע import לכל המחלקות והממשקים בחבילה, בפקודת import אחת:

```
package otherpackage;
import mypackage.*;
class Stam{
        void foo() {
                A a = new A();
                B b;
        }
}
```

חשוב לשים לב שהפקודה import איננה "מיבאת" קוד אלא פשוט מאפשרת שימוש בשם מקוצר.
אם מבצעים import לשתי חבילות הכוללות מחלקה באותו שם, עלולה להיות התנגשות שמות.
לדוגמה:

file …/myutils/A.java:

```
package myutils;
class A{…}
```

file …/otherpackage/Stam.java:

```
import mypackage.*;
import myutils.*;

class Stam{
        void foo() {
                 B b; //ok, no problem.
                 //A a; //compilation error: is it mypackage.A or myutils.A ?
                myutils.A a = new myutils.A(); // ok, no problem
                mypackage.A a1; //ok
```

```
    }
}
```

חבילת ברירת המחדל

בכל הקוד שכתבנו בקורס עד כה, לא השתמשנו כלל במנגנון החבילות. למרות זאת יכולנו לכתוב קוד כזה:

file A:

```
class A{
    static public void main(String args) {
       B b = new B();
       b.foo();
     }
}
```

file B:

```
class B{
    void foo() {…}
}
```

אם נמקם את שני הקבצים הללו באותה תיקייה, נוכל לקמפל ולהריץ את הקוד.

לפי מה שהסברנו עד כה, ל- B.foo() יש הרשאת package. נשאלת השאלה כיצד ל A הייתה הרשאה לגשת אליו? התשובה היא שכאשר לא מצוין שום שיוך לחבילה, המחלקה משויכת אוטומטית לחבילת ברירת מחדל הכוללת את כל המחלקות הלא משויכות באותה תיקייה.

בדוגמה זו, המחלקות A ו- B שתיהן שייכות לאותה חבילה (ה- default package) ולכן יש להן אפשרות להתייחס לשדות בעלי package-access.

חבילת ברירת המחדל נועדה לחסוך סרבול כאשר כותבים תוכניות בדיקה או פרוייקטים קטנים מאוד.

חבילות ועץ הקבצים

פרוייקט תוכנה אחד יכול לכלול קבצים רבים השייכים למספר חבילות. בכדי לשמור על הסדר, טבעי להשתמש בהיררכיה של מערכת הקבצים ולמקם את כל בקבצים השייכים לחבילה מסוימת בתיקיה אחת. כמו כן, כדאי למקם חבילות דומות תחת אותה תיקיה בעץ הקבצים. מנגנון החבילות של Java מכוון לארגון כזה ע"י מתן אפשרות לשמות ספריות המקבילים למיקום במערכת הקבצים.

נתבונן בדוגמה הבאה:

file Transformations.java

```
package myutils.numeric.linearalgebra;
public class Transformations {…}
```

כפי שניתן לראות, שם החבילה מורכב מכמה שמות המופרדים בנקודות. שם זה מגלה את סוף המסלול בעץ הקבצים בו נמצא הקובץ Transformations.class. המסלול יראה כך:

…/myutils/numeric/linearalgebra/Tranformations.class

תחילת המסלול, המסומנת כאן בשלוש נקודות, יכולה להיות כל מסלול הנמצא במשתנה הסביבה CLASSPATH. סוף המסלול הוא בדיוק שם החבילה, כאשר הנקודות מוחלפות בלוכסנים. כידוע, כאשר מריצים קוד באמצעות המכונה הווירטואלית, היא מחפשת את קבצי ההרצה במסלולים הנמצאים במשתנה הסביבה CLASSPATH. כאשר מחפשת המכונה את הקובץ Transformations.class, היא מחפשת אם יש המשך בצורת /myutils/numeric/linearalgebra/Tranformations.class מכל מסלול הנמצא ב- CLASSPATH.

ההקבלה בין שמות החבילות למיקום במערכת הקבצים, מאפשרת שמירה על סדר לוגי בקבצי הקוד הרבים. אם לדוגמה תהיה עוד חבילה הקשורה לחישובים נומריים אבל הפעם של קלקולוס,

טבעי יהיה לתת לה את השם myutils.numerics.calculus ולמקם אותה בתיקיה בשם calculus הנמצאת בתיקיה numeric בה נמצאת גם linearalgebra.

אם ל-Transformations יש פונקציה סטטית בשם foo(), הפעלה שלה דרך שמה המלא, תראה כך:

```
myutils.numeric.linearalgebra.Transformations.foo();
```

כאשר כל הנקודות חוץ מהאחרונה, מסמלות בעצם מסלול בעץ הקבצים.

בכדי למנוע התנגשות שמות של מחלקות, גם כאשר קוד עובר בין אנשים שונים ופרוייקטים שונים, רצוי לתת שמות ייחודיים ברמה העולמית - שמות שניתן לדעת בביטחון יחסי שאינם קיימים בקודים אחרים. בשביל להשיג ייחודיות כזו, משתמשים בגרסה של כתובות האינטרנט. אם בבעלותי domain אינטרנתי בשם orimosenzon.com (אתרים המתחילים ב - www.orimosenzon.com), אתן לחבילות שלי שמות המתחילים ב - com.orimosenzon, לדוגמה:

```
package com.orimosenzon.myutils;
....
```

מכיוון שכתובות האינטרנט הן ייחודיות, שמירה על מוסכמה זו מבטיחה ייחודיות של שמות החבילות.

הערה: המוסכמה היא לתת לחבילות שמות באותיות קטנות בלבד, אפילו אם השם מורכב מכמה מילים.

הרשאות גישה

ב - Java קיימים ארבעה מציינים של הרשאות גישה: public, protected, package-access(friendly) ו - private. הרשאת public היא המתירנית ביותר, היא מאפשרת גישה לכל קוד החפץ בכך. הרשאה זו נותנים ל - members המהווים חלק מהממשק של האובייקט לכל קוד משתמש. הרשאת protected, היא הרשאה מחמירה יותר: היא מאפשרת גישה לכל **קוד השייך לאותה חבילה** וגם לקוד של מחלקות יורשות. כל קוד אחר, לא ראשי לגשת ל - member. ההרשאה הבאה נקראת package-access או הרשאת friendly. הרשאה זו מחמירה יותר מ - protected: היא מאפשרת גישה רק לקוד השייך לאותה חבילה ולא מאפשרת גישה לכל קוד אחר (גם לא למחלקות יורשות הנמצאות בחבילות אחרות). להרשאת package-access אין מילה שמורה - זו בררת המחדל. אם לא מציינים ל member אף הרשאת גישה, אז הרשאת הגישה שלו היא package-access. ההרשאה המחמירה ביותר היא הרשאת private המאפשרת גישה רק לקוד של אותה מחלקה.

כפי שניתן לראות, מושג החבילה רלוונטי להרשאות protected ו - package-access. העובדה שמתכנני השפה בחרו בררת המחדל להיות package-access מבטאת אולי את העובדה שהרשאה זו יכולה להיות שימושית מאוד. ב - C++ בחרו מתכנני השפה את private כברירת המחדל (הרשאת package לא קיימת ב - C++). יש לשים לב, שלמרות הנוחות, הרשאת ה - package עלולה להיות מתירנית מדי מבחינת אנקפסולציה ועדיף במקרים רבים להשתמש ב - private.

חבילות ו-jar

חבילה ב - java עשויה לכלול קבצי class. רבים שחלקם קטנים מאד. בכדי שהמערכת לא תכלול הרבה קבצים קטנים, ניתן ליצור קובץ ארכיון אחד המכיל את כל קבצי החבילה ולמקם את המסלול שלו ב CLASSPATH. יצירת קובץ הארכיון תעשה ע"י שימוש ב - jar עם אופציה המונעת כווץ (הוספת דגל אפס - 0):

```
$ jar cvf0 mypackage.jar  *.class

$ CLASSPATH=$CLASSPATH:/home/oop/mypackage.jar
```

שימו לב שיש לכלול את שם קובץ הארכיון עצמו ולא רק את המסלול לתיקייה שלו.