תרגיל היכרות עם Java

דדליין: 8.6, 23:59
(איחורים בהגשת התרגיל יורדים ממכסת ימי החסד הרגילים)

בתרגיל קצר זה, נתרגל את הבסיס לשפת Java שלמדנו בכיתה, בעזרת מימוש קלאס של מספרים מרוכבים.

תזכורת - איך מריצים ג'אווה:
מומלץ לפתח את התרגיל בסביבת הפיתוח החינמית Eclipse.
יש מדריך ב-moodle עם הנחיות כיצד לקמפל פרוייקט בסביבה זו.
על מנת להריץ קוד בשרתי האוניברסיטה, יש לבצע את השלבים הבאים:

- להתחבר חיבור רגיל לנובה.
- לפתוח תיקייה חדשה כלשהי, וליצור בה קבצי java.
- לקמפל את הקבצים בעזרת הקומפיילר של ג'אווה, למשל בהנחה ששם הקלאס הוא Example:

javac Example.java

- להריץ את הקוד כך:

java Example

מומלץ לכתוב תוכנית קצרה בסגנון Hello world ולוודא שאתם מסוגלים להריץ אותה לפני שאתם ניגשים לתרגיל עצמו. הוראות הגשה לתרגיל מופיעות בהמשך.

ונעבור לתרגיל עצמו - Class המייצג מספרים מרוכבים
בתרגיל זה נממש קלאס פשוט המאפשר לבצע חישובים במספרים מרוכבים. הקלאס יקרא ComplexNumber והוא צריך לתמוך במתודות הפומביות הבאות:

ComplexNumber plus(ComplexNumber other) - returns a new ComplexNumber that is the sum of this ComplexNumber and the other ComplexNumber.

ComplexNumber minus(ComplexNumber other) - returns a new ComplexNumber that is the difference of this ComplexNumber and the other ComplexNumber.

ComplexNumber times(ComplexNumber other) - returns a new ComplexNumber that is the product of this ComplexNumber and the other ComplexNumber.

ComplexNumber divide(ComplexNumber other) - returns a new ComplexNumber that is the division of this ComplexNumber and the other ComplexNumber.

// Note: This is a constructor.

ComplexNumber(double realPart, double imaginaryPart)

double getRealPart() - returns the real part of the ComplexNumber.

double getImaginaryPart() - returns the imaginary part of the ComplexNumber.

double getRadius() - returns the radius (= absolute value) of the ComplexNumber, when represented in polar coordinates. Hint: Use the Math.sqrt() method.

double getArgument() - returns the argument (= angle, = theta) of the ComplexNumber, in radians, when represented in polar coordinates. Hint: Use the Math.atan2() method, note the order of its arguments.

boolean almostEquals(ComplexNumber other) - return whether the radius of the difference between this ComplexNumber and the other ComplexNumber is less than some constant EPSILON. Define this constant as a static final member of the ComplexNumber class, and set it to 0.001.

OPTIONAL:
static ComplexNumber fromPolarCoordinates(double radius, double argument) - return a new ComplexNumber, that has the given radius and argument.

אם אתם רק קוראים את התרגיל בפעם הראשונה לפני שאתם כותבים את הקוד, כדאי לעצור פה ולכתוב את המחלקה. אם אתם בכל זאת מתעקשים להמשיך לקרוא, עצרו פה וקחו לפחות כמה דקות לתכנן את המחלקה בראש בקווים כלליים: איך כל מתודה ממומשת, ואיזה data members יש למחלקה.

הוראות הגשה:

הטסט האוטומטי של התרגיל נמצא פה:

~nimrodav/java_prep_ex/run_test.sh

יש ליצור תחת תיקיית c_lab שלכם תיקיה בשם java_prep_ex, לשים בה את הקוד שלכם, להעתיק את Main.java מתיקיית הטסט לתיקיה שלכם, ולקמפל את הקוד בעזרת (למשל)

javac *.java

במועד ההגשה, שתי הפקודות הבאות צריכות להסתיים בהצלחה ולא להדפיס כלום למסך:

```
cd ~/c_lab/java_prep_ex
~nimrodav/java_prep_ex/run_test.sh
```

התרגיל יבדק יחד עם תרגיל ג'אווה המלא, ויהווה 10% מהציון של התרגיל המלא. תרגילים סבירים צפויים לקבל את מלוא 10% הנקודות.

כתבתם את המחלקה ובדקתם שהקוד נראה תקין? מצוין. עכשיו נראה דוגמה קלאסית להפרדה בין מימוש וממשק.
אולי שמתם לב שהממשק של המחלקה לא מעיד על איזה data members יש בה. סביר שבחרתם לממש את המחלקה עם שני data members: הקואורדינטה הממשית של המספר, והקואורדינטה המדומה שלו. אבל זאת בחירת מימוש, ואפשר לשנות אותה מתי שרוצים מבלי לשנות את הממשק!
ועכשיו נעשה את זה: החליפו את המימוש כך שה-data members של המחלקה הם הקואורדינטות הפולאריות של המספר המרוכב (R ו-theta). אם מראש מימשתם באמצעות קואורדינטות פולאריות, עשו את ההיפך: החליפו את המימוש לקואורדינטות קרטזיות. (אין צורך להגיש את המימוש הנוסף.)
<u>שימו לב שהממשק של המחלקה, כלומר הארגומנטים וטיפוסי החזרה של המתודות הפומביות, לא השתנה כלל עקב שינוי המימוש!</u> (אם הייתם צריכים לשנות ולו אות אחת בשורת החתימה של מתודה פומבית, משהו לא בסדר).
ובזאת סיימנו עם התרגיל. בתקווה אתם מרגישים כעת מתורגלים בכתיבת קוד קצר בג'אווה, ואף ראיתם את הכח של הפרדה בין ממשק ומימוש. ולסיום, כמה הערות:
ראשית, כדאי לציין שיש עוד כמה בחירות מימוש אפשריות, למשל להחזיק במחלקה גם קואורדינטות קרטזיות וגם קואורדינטות פולאריות, לבצע כל חישוב בעזרת הקואורדינטות הרלבנטיות, ואז לעדכן את הקואורדינטות השניות. אפשר גם לחשוב על מימוש "מתחכם" שמחזיק בכל רגע נתון רק זוג אחד של קואורדינטות, לפי מה שנוח לו כרגע, מתוך מטרה לנסות לבצע כמה שפחות חישובים. שימו לב שגם בחירות המימוש הללו לא גוררות שינוי בממשק.
שנית, אולי הפריע לכם בעיניים שלמרות שאנחנו עוסקים בפעולות חשבון כמו חיבור, כפל, וכו', אנחנו נדרשים לקרוא לפונקציה plus משיקולי תחביר של השפה. תיאורטית, היינו ממש שמחים אם היה אפשרי לכתוב את הקוד הבא:

```
ComplexNumber a = new ComplexNumber(1, 2);
ComplexNumber b = new ComplexNumber(3, 4);
ComplexNumber c = a + b;
```

כאשר פעולת החיבור מיוצגת בעזרת הסימן ההגיוני, +, ומאחורי הקלעים רצה מתודה שאנחנו כתבנו שמממשת חיבור בין שני instances של ComplexNumber.
לצערנו, שפת Java לא תומכת בקונספט הזה, שנקרא operator overloading. שפת Python כן תומכת בקונספט הזה. בנוסף לכך, שפת C++ היא מעין הרחבה של שפת C, שאף היא תומכת בקונספט הזה.