

פתרון מבחן בתוכנה 1, 2 ביולי 2006

הציון הסופי נקבע בצורה הבאה. יהי x הציון הגולמי על המבחן, ויהי y מספר התרגילים שהוגשו וקיבלו ציון עובר. הציון הסופי הוא

$$100 - 0.666 * (100 - (x + y))$$

משמעות הנוסחה היא שכיווצנו את הפער בין הציון הגולמי $x + y$ ובין 100 בפקטור של שני שליש, כדי לקרב את הציונים ל-100.

אי הגשת תרגילים הביאה לסימון "לא עמד/ה בחובות הקורס" בגיליון הציונים. זה לא נחשב נכשל וזה לא נחשב עובר. תלמיד או תלמידה שקיבלו סימון כזה למרות שהגישו את התרגילים מתבקשים לפנות אלי (סיון) לבירור הסיבה שאין לנו ציונים עבור חלק מהתרגילים.

במקרים מסויימים תיארונו טעויות בגוף המבחן במילים, ולעיתים השתמשנו בקוד (A, B, וכדומה) לציון שגיאה שחזרה על עצמה.

שאלה 1

סעיף א:

הפתרון מבוסס על יצירת בנאי ל-`Policy` שישמור התייחסות ל-`this` במבנה נתונים כלשהו, כאשר למבנה הנתונים עצמו תהיה התייחסות משדה מחלקה. מבנה הנתונים יכול להיות רשימה מקושרת, מערך, או `Collection` כלשהו מהספרייה. כמו כן יש צורך בשירות מחלקה שיחזיר התייחסות למבנה הנתונים, או שיחזיר איטרטור שעובר על איבריו.

השירות `Dispatch.dispatch()` צריך להכיל לולאה אינסופית. בכל מעבר בלולאה צריך לעבור על כל איברי מבנה הנתונים ששמור ב-`Policy`, להפעיל את השירות `testAndRespond` של כל `Policy`, ולהמתין 10 מילישניות.

קודים לשגיאות והפחתת הנקודות בגיין:

A. שימוש ב-`Policy` כאילו היה `Collection`. -4

B. מערך בגודל קבוע שלא יספיק אם יש הרבה עצמים מטיפוס `Policy`. -1

C. המתנה לאחר כל קריאה ל-`testAndRespond` (ולא המתנה רק אחרי שקוראים לכל השירותים הללו). -1

סעיף ב:

הקריאות ל-`testAndRespond` בתוכנית הנתונה גורמות להקצאות נתונים בגלל שהחיישן מחזיר עצם מטיפוס `Double` ומכיון ש-`Actuator.act` מצפה לקבל ארגומנט מטיפוס `Boolean`. למרות שמעבירים את הקבוע `true`, למשל, נוצר `Boolean`. ה-`Boolean` וה-`Double` הם עצמים שמוקצים על ה-`heap`, להבדיל ממשתנים מטיפוס פרימיטיבי שמוקצים על המחסנית.

A. התשובה ציינה שיש הקצאה תכופה של `Boolean`, `Double`, אבל שאוסף הזבל לא מופעל. -2

B. הקצאה תכופה של `Double` אבל בלי להזכיר `Boolean` או להיפך. -1

C. נוצרים רק משתנים מטיפוס פרימיטיבי. -3

סעיף ג:

פתרונות לחלק א' שמבוססים על איטרטור גורמים להקצאה של עצם איטרטור בכל מעבר בלולאה של `dispatch`. פתרונות שמבוססים על מעבר על רשימה מקושרת או על מערך יכולים לעבור על מבנה הנתונים תוך שימוש אך ורק במשתנים של `dispatch` שנשמרים (פעם אחת) על המחסנית.

A. שימוש באיטרטור והצהרה שגויה שלא מוקצה זיכרון בגין האיטרטור. 2-

סעיף ד:

מערך הקלט צריך לכלול שני איברים לפחות. שני האיברים (מחרוזות) צריכים להכיל ייצוג של מספרים. המספר הראשון צריך להיות קטן מהשני.

A, B, C. חוסר באחד משלושת הפסוקים שלמעלה, בהתאמה. 2- על כל חוסר.

סעיף ה:

השירות main מיועד להיות מופעל על ידי בני אדם (דרך מערכת ההפעלה), ולכן יש לצפות לטעויות אנוש בקריאה לשירות. לכן עדיף שלא יהיו לשירות תנאי קדם כלל. השינויים הדרושים הם בדיקה של תנאי הקדם ויציאה מהתוכנית עם הודעת שגיאה מתאימה אם הם לא מתקיימים.

סעיף ו:

יש מספר אפשרויות לפתרון. אפשרות אלגנטית אחת היא לממש את המחלקה שתייצג את המתג החכם כך שתממש גם מנשק של `Actuator<Boolean>` וגם מנשק של `Sensor<Boolean>`. כשנשתמש בעצם כזה כחיישן, הוא ידווח האם הפעולה האחרונה של המתג הצליחה או נכשלה. זה מצריך שינוי בלקוח שרוצה לבדוק הצלחה/כישלון של המתג החכם, אבל לא דרושים שינויים נוספים.

פתרון אפשרי אחר הם להוסיף למחלקה שמייצגת את המתג החכם שאילתה שמחזירה האם ההפעלה האחרונה הצליחה. גם זה מצריך רק שינוי בלקוחות. פתרון שלישי הוא להודיע על חריג מתוך `act` אם ההפעלה נכשלת, אבל זה מצריך שינוי גם במנשק `Actuator<T>`. זה חסרון גדול. יש גם שיקולים של יעילות ושל הכרח לטפל ב-`checked exception`.

A. רק שיטה אחרת. 3-

B. `Act` יחזיר ערך (זה יפעל, אם משנים את `Actuator<T>`, אבל זה מעררב שאילתה ופקודה).

1

C. החזרה של ערך הצלחה/כישלון בלי לציין איך הוא מוחזר (ערך מוחזר משירות, חריג, וכו').

2

D. שימוש של לקוח בשדה מופע בלי שאילתה 1-

חלק ז:

הפתרון אחד (הסביר ביותר) מגדיר מחלקה פנימית שמייצגת מתג "הפוך", שהבנאי שלו מקבל מתג רגיל ושומר התייחסות אליו בשדה מופע. כאשר מפעילים `act(true)` על המתג ההפוך הוא קורא לשירות `act(false)` של המתג שהועבר לבנאי, ובאופן דומה עם `act(false)`. הבנאי של `HysteresisDownController` בונה כזה מתג מתוך המתג הנתון, ואז בונה `HysteresisController` רגיל עם המתג ההפוך.

פתרון דומה מבוסס על חיישן הפוך, שמבצע טרנספורמציה על קריאות של חיישן נתון, למשל הכפלה שלהם ב-1. זה גורם לערכים של החיישן ההפוך לרדת כאשר החיישן האמיתי עולה ולעלות כשהאמיתי יורד.

בנייה פשוטה של `HysteresisController` עם החלפה של ערכי הסף הנמוך והגבוה לא עובדת.

A. שכפול של הקוד הקיים עם החלפת כיוון של בדיקות <, >. 5-

B. רעיון נכון אבל מימוש לא נכון (למשל שימוש לא נכון במתג הנתון מתוך המתג ההפוך). 2-

C. הפתרון לא משתמש כלל בחיישן ו/או ב-`actuator` הנתון (זה גם מצריך שכפול קוד, של החיישן או המתג).

שאלה 2

- סעיף 1. הקוד תקין, הערך של d הוא אינסוף שהוא גדול מ-1 ולכן יודפס X. קטע קוד דומה הופיע בקוד שהיה צריך להוכיח נכונות שלו באחד התרגילים.
- סעיף 2. הקוד לא תקין, ההגדרה int I היא של משתנה בשירות (לא של שדה), מכיון שהשורה הבאה קוראת לשירות. לפני שאפשר להשתמש במשתנה בשירות חייבים לבצע לו השמה. אין איתחול אוטומטי כמו של שדות.
- סעיף 3. הקוד תקין אם כי לא שגרתי. יודפס B, ומכיון שאין break אחרי ההדפסה יבוצעו גם הפקודות הבאות, כך שיודפס BCXXX.
- סעיף 4. הקוד תקין. מערך הקלט יאותחל על ידי מערכת ההפעלה ל-hello באיבר הראשון של המערך ול-world באיבר השני. אורך המערך יהיה 2, ולכן האיבר השלישי, arr[2], לא קיים, והקוד ייצר בזמן ריצה חריג שמדווח על אינדקס שחורג ממימדי המערך.
- סעיף 5. הקוד תקין ומדפיס 4. ההצהרה final מקבעת את הארגומנט a: לא ניתן לבצע לו השמה (לשנות את ההתייחסות a כך שתתייחס לעצם אחר), אבל ניתן לשנות את השדות של העצם a-ש מתייחס אליו.
- סעיף 6. הקוד לא תקין, מכיון ש-Inner היא מחלקה פרטית של Outer, אז אי אפשר להתייחס לטיפוס Outer.Inner מחוץ ל-Outer. לכן השירות main, שהוא מחוץ ל-Outer, לא יכול להשתמש בטיפוס Outer.Inner. הטיפוס אינו מוכר ולכן ההכרזה על inner אינה תקינה.

שאלה 3

- סעיף 1. לא תקין, לא ניתן לדרוס שירות public עם שירות protected.
- סעיף 2. לא תקין, אי אפשר להודיע על חריג כי בהכרזה על השירות אין פסוק throws מתאים. החריג IOException הוא checked exception.
- סעיף 3. תקין, מעמיס את f. יודפס in B ואחר כך in A.
- סעיף 4. תקין, מעמיס את f, אבל לא קוראים לשירות הזה כלל. יודפס in A ושוב in A.

שאלה 4

סעיף א:

תשובה מלאה:

הקוד מקצה שני מערכים של מערכים של double בגודל משתנה. במערך הראשון, L, גודל תתי המערכים עולה מ-1 ל-10 ושמש בתרגיל להחזקת מטריצה משולשית תחתונה. במערך השני, U, גודל תתי המערכים יורד מ-10 ל-1 ושמש בתרגיל להחזקת מטריצה משולשית עליונה.

תשובה שהתקבלה צריכה לכלול את הפרטים הבאים:
 L - מטריצה משולשית עליונה, U מטריצה משולשית תחתונה.
 - מערכים בגודל משתנה (ציור התקבל גם כן)

קודי שגיאה:

סימון קוד	הסבר	נקודות
41A	אין פירוט של הצורך (אי התייחסות למטריצות משולשית עליונה ותחתונה)	1-
41B	אי התייחסות לכך שזהו מערך של מערכים בגודל משתנה	1-

סעיף ב:

תשובה מלאה :

הצמתים מייצגים את מצב המשחק, דהיינו מצב הלוח + מיהו השחקן הבא בתור לשחק. השרש הוא המצב ההתחלתי של המשחק. כל קשת מייצגת מהלך חוקי של השחקן הבא בתור והיא מובילה לצומת שמייצג את מצב המשחק לאחר מהלך זה.

תשובות שהתקבלו : מצב המשחק, מצב הלוח.

נקודות	הסבר	סימון קוד
0	מהלך	42A
3-	ציון – כמה שחקן מסויים קרוב לנצחון	42B
3-	תשובה לא מדוייקת/ מעורפלת - "מצב" (ללא התייחסות של מה), כל המהלכים האפשריים של השחקן הנוכחי, ...	42C

סעיף ג :

תשובה מלאה : Big Endian מגדיר את אופן הייצוג של מספר ע"י מערך של בתים : הבית "המשמעותי" (דהיינו הבית עם הסיביות המשמעותיות) מופיע ראשון, ואחריו מופיעים שאר הבתים בסדר יורד לפי המשמעות שלהם (הבית עם הסיביות הכי משמעותיות יופיע אחרון).

תשובות שהתקבלו :

- שמירת נתונים בתוך קובץ כאשר ה- msb בהתחלה.

סימון קוד	הסבר	נקודות
43A	תשובה לא נכונה / מדוייקת שכוללת התייחסות לסדר.	2-
43C	תשובה לא מדוייקת שכוללת התייחסות לסדר ו"משמעות" (צמידים של בתים, התייחסות לסיביות או ל- nibbles ולא לבתים)	1-

סעיף ד :

תשובות שהתקבלו :

- שרת HTTP לא רץ
- בעיה בגישה לשרת HTTP
- כתובת לא נכונה של ה- servlet ב- Browser
- בעיה ב- deployment לשרת (הקבצים לא במיקום המתאים)
- עדכון קבצים ללא הרצה מחדש של השרת

סימון קוד	הסבר	נקודות
44A	ציון בעיה בקוד בריצה של הקוד של ה- servlet (למעשה ההתנהגות אכן תואמת) 44A1 – בעיה ב- doGet/dopost/service 44A2 – בעיה ב- request 44A3 – input לא חוקי מהמשתמש	0 (-5)