2020 / 01 / 28
כ"ד באב תשע"ח

האוניברסיטה העברית בירושלים

ביה"ס להנדסה ומדעי המחשב ע"ש רחל וסלים בנין

מבוא לתכנות מונחה עצמים (67125)

(מועד א')

מרצה: יואב קן-תור

מתרגלים: ג'ונה הריס, אלעד גרשון

צארים: שחר רסיסי , רועי פרידמן

תעודת זהות של הסטודנט/ית: _____

הוראות:

- משך המבחן הינו **3 שעות**
- המבחן הינו עם חומר סגור. אין להשתמש בכל חומר עזר או אמצעי חישוב שהוא, לרבות מחשבון
- למבחן שני חלקים:
  - חלק א' – כולל 6 שאלות, מהן יילקחו 5 השאלות בעלות הניקוד הגבוה ביותר
  - חלק ב' – כולל 2 שאלות קוד גדולות

יש לענות על כל שאלות המבחן במחברת הבחינה.

שימו לב לנקודות הבאות:

- "Less is more" – רשמו תשובה קצרה ומדויקת, שלא תשאיר מקום לספק שאכן הבנת את החומר
- ייתכן שהקוד המוגש יעבוד, אבל למבחן בתכנות מונחה עצמים זה לא מספיק. הקוד צריך להיות קריא ומתוכנן היטב. תעד את הקוד (אפשר בעברית) רק אם יש חשש שמישהו לא יבין אותו
- זכור כי בעיצוב אין תשובה אחת נכונה, אבל בהחלט יש תשובות שהן לא נכונות. נקודות מלאות יתקבלו רק באם השתמשת באחת מהדרכים המתאימות ביותר
- יש לתכנת בשפת JAVA
- אנו ממליצים לקרוא את כל המבחן מתחילתו עד סופו לפני תחילת הפתרון
- המבחן כתוב בלשון זכר, אך מיועד לכלל הסטודנטיות והסטודנטים הלוקחים את הקורס במידה שווה

בהצלחה!

חלק א'

- כולל 6 שאלות, מהן יילקחו 5 השאלות בעלות הניקוד הגבוה ביותר. כל שאלה שווה, לכל היותר, 10 נקודות. הקפידו לנמק כאשר התבקשתם.
- חלק מן השאלות כוללות מספר סעיפים, אשר מופרדים לתיבות שונות במודל ומסומנים בהתאם (למשל, אם שאלה 1 כוללת שני סעיפים הם יסומנו כ"שאלה 1.1" ו"שאלה 1.2"). הקפידו לענות על כל חלקי השאלות חלקי השאלה הוא לנוחות הפתרון בלבד.
- בשאלות הכוללות כתיבת קוד, הקפידו לכתוב קוד **קצר, פשוט ומובן ב-JAVA המשתמש בכלים המתאימים ביותר**. רק מימושים כאלו יקבלו ניקוד מלא.

שאלה 1

כדי ליצור אובייקט של מחלקה מקוננת (nested inner class) **[חייבים\לא חייבים]** ליצור אובייקט מהמחלקה העוטפת אותה (outer class)
מחלקה מקוננת סטטית (nested static class) **[יכולה\לא יכולה]** לגשת לכל המשתנים של המחלקה העוטפת אותה (פרטיים, מוגנים, סטטיים, לא סטטיים וכו')
מחלקה מקוננת **[יכולה\לא יכולה]** לגשת לכל המשתנים של המחלקה העוטפת אותה (פרטיים, מוגנים, סטטיים, לא סטטיים וכו')
מחלקה עוטפת **[יכולה\לא יכולה]** לגשת לכל המשתנים שהוגדרו בתוך מחלקה שמקוננת בה
מחלקה מקוננת **[יכולה\לא יכולה]** להיות מוגדרת כפרטית (private)

שאלה 2

תארו והסבירו את תבניות העיצוב Decoration ו- Delegation ותנו דוגמא לשימוש נכון באחת מהן (ניתן להסביר על סמך תרגיל שמימשתם בקורס).

שאלה 3

הסבר ותאר את מנגנון הגנריות (Generics) ומדוע משתמשים בו באוספים (Collections).

שאלה 4

הסבירו בקצרה מה המטרה של unit testing ותארו בקצרה את המשמעות של **3 מהאנוטציות** הבאות מ-JUnit:

Before	BeforeClass	Test	After	AfterClass	Test(expected =ExceptionA)

שאלה 5

ממשו מתודה המקבלת מספר (int) ובודקת האם הוא מייצג מספר בינארי (מספר שכל ספרותיו הן 0 ו-1). על החתימה של המתודה להיות:

```
public static boolean isBinary(int num);
```

<u>הערה</u>: ניתן להניח שהמספר הנתון אינו שלילי ונמצא בטווח המתאים ל-int בג'אווה.

דוגמאות:

- הרצת המתודה על 1234512 תחזיר false
- הרצת המתודה על 110101 תחזיר true
- הרצת המתודה על 10000 תחזיר true
- הרצת המתודה על 100110002 תחזיר false
- הרצת המתודה על 0 תחזיר true

שאלה 6

ממשו מתודה המקבלת מחרוזת (String) ובודקת האם סדר הסוגריים במחרוזת תקין. על החתימה של המתודה להיות:

```
public static boolean parOrder(String line);
```

<u>הערה</u>: ניתן להניח שאין תו יציאה (escape character) במחרוזת הנתונה (כלומר, אין דברים כמו '\n' במחרוזת).

<u>הגדרה</u>: סוגריים מורכבות מתו (character) פתיחה ותו סגירה. תווי הפתיחה האפשריים הם "(", "[" או "{" ותווי הסגירה המתאימים להם הם ")", "]" ו-"}", בהתאם. מחרוזת תקרא תקינה (מבחינת סוגריים) אם מתקיים:

1. אם קיים תו פתיחה, אז קיים תו סגירה מתאים (ולהפך) - כלומר, אם אין תו פתיחה, גם לא יהיה תו סגירה (ולהפך)
2. בין תו הפתיחה של סוגריים לתו הסגירה של אותם הסוגריים תופיע תת-מחרוזת אשר בעצמה הינה תקינה מבחינת סוגריים (שימו לב שזו יכולה להיות המחרוזת הריקה). כלומר, יכול להופיע שילוב של אותיות ומספרים (תווים מהסוג [a-zA-Z1-9] וכן תווים מיוחדים דוגמת #,%,&) וכל תת-מחרוזת שבעצמה תקינה מבחינת סוגריים

דוגמאות:

- הרצת המתודה על "no parentheses" תחזיר true
- הרצת המתודה על "good parentheses ()" תחזיר true
- הרצת המתודה על "({[]})" תחזיר true
- הרצת המתודה על "({[])}" תחזיר false
- הרצת המתודה על "(()" תחזיר false

חלק ב'

- קראו בעיון את שתי השאלות הבאות. קראו כל שאלה עד הסוף לפני תחילת הפתרון
- לשאלות יכולות להיות מספר סעיפים ותתי-סעיפים. החלוקה לסעיפים נועדה לנוחות הפתרון בלבד
- שאלה ללא נימוק תאבד נקודות
- כאשר אתם כותבים נימוק הקפידו לפרט באילו עקרונות ותבניות עיצוב (design patterns) בחרתם להשתמש ומדוע
- הקפידו על עיצוב יעיל
- בכל פעם שהוגדרה מתודה למימושכם, אם חתימתה חסרה, עליכם להשלים אותה לפי שיקולכם
- ניתן להשתמש בקוד Java של מחלקות שלמדנו עליהן (לדוגמא הפונקציה sort של Collections)

שאלה 1

רקע

חברת התעופה באג-אין אירליינס החליטה להיעזר בכם לפיתוח מערכת ה-logging של מטוס הבאג-פורס-וואן החדש שלה. מערכת logging היא מערכת לתיעוד אירועים ומצבים שונים. התיעודים יכולים להתבצע בזמנים שונים במהלך הטיסה (מהמראה עד נחיתה). התיעוד נשמר בזמן אמת בכמה מקומות. מערכת זו מאפשרת לעקוב אחר מצב המטוס, לוודא האם המערכות תקינות, לגלות תקלות שונות ועוד.

כל מערכת logging כוללת את החלקים הבאים:

הודעות תיעוד	כל הודעת תיעוד log_message ברמה log_level צריכה להירשם ליעד (בהנחה והיעד תומך בסוג תיעוד זו) בפורמט הבא: **YYYY-MM-DD HH:mm log_level: log_message**
רמת תיעוד (log-level)	ישנם 4 סוגי תיעוד: 1. **DEBUG**. הודעות תחזוקה 2. **INFO**. מידע אודות אירועים כלליים בזמן הטיסה 3. **WARNING**. אזהרות לגבי מצב המערכות במטוס 4. **ERROR**. מידע על תקלות במערכות המטוס
יעד (Handler)	יעד אליו נכתבים תיעודים. לכל Handler יש ערך log-level מינימלי. Handler יכתוב תיעודים בעלי log-level גבוה (או שווה) לערך זה. יעדים אפשריים הם למשל "מסך הטייס" או "הקופסה השחורה"
לוגר (Logger)	האובייקט דרכו מבצעים תיעודים. אובייקט זה מכיל מספר יעדים שונים ומעביר הודעות לוג אליהם.

הבהרות:

- סדר חשיבות ה-log-level הינו ע"פ המספור לעיל, מהקטן (זניח) לגדול (חשוב ביותר)
- בשאלה זו נספק לכם API בו תוכלו להיעזר לצורך מימוש היעדים **"מסך הטייס" ו"הקופסה השחורה"**. עם זאת, זכרו כי לכל Logger יכול להיות **מספר יעדים גדול יותר**
- לכל יעד יכול להיות log level מינימלי **שונה**
- **YYYY-MM-DD HH:mm** הם התאריך והשעה של זמן התיעוד. השתמשו במחלקת העזר הנתונה TimeUtils כדי לקבל את התאריך והשעה בפורמט הרצוי (ראו API בהמשך)
- אובייקט Handler אחד יכול להיות במספר Loggerים

דוגמאות:

סדר סוגי התיעוד

להזכירכם סדר ה log-level הוא DEBUG < INFO < WARNING < ERROR:

- Handler שהוגדר עם log-level מינימלי WARNING יכתוב רק לוגים ברמות WARNING,ERROR
- Handler שהוגדר עם log-level מינימלי ERROR יכתוב רק לוגים ברמה ERROR
- Handler שהוגדר עם log-level מינימלי DEBUG יכתוב לוגים בכל הרמות

פלט אפשרי ממערכת logging של מטוס

כפי שניתן לראות, המערכת מספקת מידע שימושי לגבי ההתנהגות והמצב של המטוס בזמנים שונים.

2019-12-24 23:45 INFO: Preparing to take-off
2019-12-24 23:58 INFO: Starting take-off
2019-12-24 23:58 DEBUG: Engine temperature before take-off: 140
2019-12-25 00:14 INFO: Take-off complete
2019-12-25 00:28 WARNING: High engine temperature: 190
2019-12-25 01:09 ERROR: Passenger multimedia isn't working
2019-12-25 2:37 INFO: Preparing to land
2019-12-25 2:42 INFO: land complete

קוד עזר

באפשרותכם להשתמש במחלקות והטיפוסים הנתונים הבאים - אין צורך לממש אותם

public class TimeUtils	
public static String now()	מתודה המחזירה את הזמן הנוכחי בפורמט "YYYY-MM-DD HH:mm"

public class BlackBox	
public static BlackBox getBlackBox()	מתודה המחזירה את הקופסה השחורה של המטוס
public void write(String str)	מתודה הכותבת את המחרוזת הנתונה לקופסה השחורה

public enum LogLevel {DEBUG, INFO, WARNING, ERROR}

תזכורת: לכל Enum מוגדרת באופן אוטומטי המתודה compareTo המשווה את האובייקטים מסוג ה Enum ע"פ סדר הגדרתם

סעיף א'

נתון לכם ה- API של המחלקה Logger, תכננו וממשו את המחלקה, את הטיפוס Handler ואת היעדים "מסך הטייס" ו"קופסה שחורה".

public class Logger	
public Logger()	יצירת Logger חדש ללא Handlers
public void addHandler(Handler handler)	הוספת Handler חדש ל Logger זה
public void log(LogLevel level, String msg)	תיעוד ההודעה msg ברמה level. התיעוד מועבר לכל ה- Handlers ב- Logger זה

קטע הקוד הבא מדגים שימוש במערכת הרצויה וצריך לעבוד עם הקוד שמימשתם:

```
public static void main(String[] args) {
   Logger logger = new Logger();
   Handler blackBoxHandler = <create Handler with log-level DEBUG, writes to the black-box>
   Handler monitorHandler = <create Handler with log-level INFO, writes to the pilot's monitor>
   logger.addHandler(blackBoxHandler);
   logger.addHandler(monitorHandler);
   logger.log(LogLevel.DEBUG, "my debug msg");
   logger.log(LogLevel.INFO, "my info msg");
   logger.log(LogLevel.WARNING, "my warning msg");
   logger.log(LogLevel.ERROR, "my error msg");
}
```

Black box output (log-level=DEBUG)	Pilot monitor output (log-level=INFO)
2019-12-16 10:27 DEBUG: my debug msg 2019-12-16 10:27 INFO: my info msg 2019-12-16 10:27 WARNING: my warning msg 2019-12-16 10:27 ERROR: my error msg	2019-12-16 10:27 INFO: my info msg 2019-12-16 10:27 WARNING: my warning msg 2019-12-16 10:27 ERROR: my error msg

הבהרות:

- כתיבה למסך הטייס מתבצעת באמצעות מתודת ההדפסה של ג'אווה - System.out.print.
- כתיבה לקופסה השחורה מתבצעת באמצעות המחלקה **BlackBox** שתוארה לעיל.

סעיף ב'

בחברת התעופה הבינו כי ישנם Handlers אשר הכתיבה אליהם דחופה יותר מ- Handlers אחרים. לדוגמה, היינו רוצים שכתיבה למסך הטייס תתבצע לפני כתיבה לקופסה השחורה, ללא קשר לסדר הוספת ה- Handlers.
לשם כך החליטו להוסיף מנגנון ל- Logger אשר יאפשר לציין את העדיפות (priority) של כל Handler:

- העדיפות של Handler מיוצגת ע"י מספר שלם כלשהו (int).
- כאשר מבצעים פעולת log, ה- Logger צריך לכתוב לכל ה- Handlers שלו ע"פ העדיפויות שלהם בסדר יורד.

שנו את המתודה addHandler במחלקה Logger כך שתהיה עם החתימה הבאה:

הוספת Handler חדש ל Logger זה עם עדיפות **priority**	public void addHandler(Handler handler, int priority)

ממשו במחלקה Logger את השינויים הנדרשים כך שהמתודה log תתנהג בהתאם לדרישות, כלומר תכתוב ל- Handlers ע"פ הסדר הנדרש.

הבהרות:

- שימו לב - אין קשר בין log-level לבין priority. כלומר, log-level **לא** משפיע על ה-priority (ולהפך)
- לאחר השינוי בסעיף זה, קטע קוד ה-main מסעיף קודם יצטרך כמובן להשתנות כדי להתאים ל-API החדש של addHandler. התעלמו מכך
- ניתן להניח כי לא יוסיפו Handler יותר מפעם אחת ל- Logger כלשהו
- סדר יורד הינו מהגדול לקטן

סעיף ג'

הסבירו כיצד ניתן להרחיב את הקוד שכתבתם כך שיאפשר שימוש ב- Handlers חדשים נוספים (למשל Handler שכותב למגדל הפיקוח). תארו את הטיפוסים שנצטרך לשנות/להוסיף.
האם המימוש שלכם מקיים את עקרון ה- open-closed?
אין צורך לממש קוד בסעיף זה

שאלה 2

רקע

מדינת ישראל רוצה לבנות מערכת תחבורה יעילה, אשר תורכב ממספר 'צורות' (Modes). אמצעי תחבורה אלו יוכלו לנוע בתנאי שטח שונים ואף באוויר. לכן, פנו ללא אחר מאשר פרופ' ריק. לאחר שזה דיבר עם שר התחבורה, הם הסכימו על רעיון חדש: רובוריקים אשר יוכלו לעבור בין מצבי תנועה. הרובוריקים יוכלו להסיע נוסעים ולשנות את צורתם על פי הצורך.

שימו לב: בשאלה זו נתעסק עם שתי צורות בלבד, אך העיצוב שתבחרו צריך לאפשר הוספה של צורות נוספות בקלות.

כידוע לכם, מערכת תחבורה הינה מלאת תקלות שונות. לכן, פרופ' ריק דורש שתבנו עץ ירושה של חריגות (**Exceptions**), ושתוסיפו חריגות במקומות המתאימים לכך.

סעיף א'

הטיפוס רובוריק (**RoboRick**) מוגדר כך:

public void changeMode(_____ newMode)	**מתודה המחליפה צורה אם יש מעל 50 אנרגריק.** מפחית 50 אנרגריק מהאנרגיה של הרובוריק ומשנה את צורתו לצורה החדשה newMode
public int getPassengers()	מתודה המחזירה את כמות הנוסעים ברובוריק
public void setPasngers(int passengers)	מתודה שמעדכנת את מספר הנוסעים (אין הגבלה על מספר הנוסעים)
public int getEnergRick()	מתודה המחזירה את מידת האנרגריק של הרובוריק

המחלקה מצב הנסיעה (**DriveMode**) מוגדרת כך:

public DriveMode()	בנאי המחלקה, מכניס 4 גלגלים
public void addWheels(int wheels)	מתודה המעדכנת את מספר הגלגלים ע"י ניסיון להוסיף כמות wheels לכמות הקיימת. **רובוריק לא יכול להכיל יותר מ-5 גלגלים**
public void removeWheels(int wheels)	מתודה המורידה כמות גלגלים נתונה. **ניתן להסיר רק מספר גלגלים קטן מהקיים.**

המחלקה מצב טיסה (**FlightMode**) היורשת מהמחלקה FlightModeBase מוגדר כך:

public FlightMode()	בנאי מחלקה
public void perpareForFlight()	מתודה המכינה את המטוס לטיסה. קיים לה מימוש גם במחלקה FlightModeBase. הפעלתה יקרה מאד במשאבים.
Public boolean isReadyForFlight()	מתודה הבודקת האם המטוס מוכן לטיסה.

ממשו את הטיפוסים **RoboRick**, **FlightMode**, **DriveMode** על כל פעולותיהם והוסיפו בנאים כנדרש. **השלימו את חתימות הפונקציה היכן שנדרש.**

להזכירכם, הוסיפו חריגות בכל מקום הדורש חריגה ושנו את חתימות הפונקציות בהתאם.

הבהרות

- הניחו כי מספר הנוסעים וכמות הגלגלים המוכנסת היא חיובית
- הרובוריק מתחיל במצב נסיעה (DriveMode) עם אנרגריק של 200
- המחלקה RoboRick אינה מאפשרת הוספת אנרגריק
- המחלקה FlightModeBase נתונה לכם ואין צורך לממשה. בין היתר, היא מכילה בנאי ברירת מחדל (default constructor) ואת המתודה perpareForFlight, ללא ארגומנטים

סעיף ב'

עכשיו הגיע הזמן לתזוזה של הרובוריקים! פרופ' ריק הבין שצריך to get swifty (לזוז) ולכן הוסיף עבורכם (ועבור מורטי) את הפעולות הבאות במחלקת **GetSwifty** (*אין צורך לממש*)

static void rickMotion(float x, float y, RoboRick rivka)	שיטה אשר מזיזה את הרובוריק לנקודת הציון הנתונה. **רובוריק מסוגל לנוע פחות מ-100 ק"מ בכל הפעלה.**
static float rickDistance(float x, float y, RoboRick rivka)	שיטה המחזירה מרחק בקילומטרים בין נקודת ציון למיקום הנוכחי של הרובוריק.

עליכם להשתמש בשיטות הללו ולממש את הפונקציונליות הבאה ב-RoboRick:

void move(float x, float y)	מתודה אשר מזיזה את הרובוריק לנקודת הציון הנתונה. **במצב נסיעה הרכב יזוז רק אם יש בו לפחות 4 גלגלים. במצב טיסה הרובוריק יזוז רק אם הוא הוכן לטיסה לפני כן.**

<u>שימו לב:</u> הרובוריקים עצמם לא מטפלים במיקומם. כל תזוזה ושינוי מקום של רובוריק מתנהל בתוך המחלקה GetSwifty, שמומשה עבורכם.

להזכירכם, הוסיפו חריגות בכל מקום הדורש חריגה ושנו את חתימות הפונקציות בהתאם.

סעיף ג'

אחרי שפרופ' ריק סיים לבנות את הרובוריק שלו, הוא תהה: "מה יותר טוב מרובוריק אחד?" וענה לעצמו: "הרבה רובוריקים!". לכן, הוא בנה רשימה מקושרת של רובוריקים. לצערו, הרשימה הייתה מבולגנת במצבים ובנוסעים. לכן רצה ריק למיינם לפי סדר מסוים.

הסדר הטבעי עליו חשב הוא שכל רובוריק תלוי במצב (mode) בו הוא נמצא. לשם כך נגדיר את מקדם המצב, ונקבע כי הוא שווה ל-3 אם הרובוריק במצב טיסה, ול-2 אם הרובוריק במצב נסיעה. כעת נגדיר את ערך הרובוריק, דרך מקדם המצב כפול מספר הנוסעים. לדוגמא:

- ערך רובוריק במצב טיסה עם שני נוסעים יהיה 3*2=6
- ערך רובוריק במצב נסיעה עם ארבעה נוסעים יהיה 2*4=8

לעומתו, שר התחבורה החדש ביצה-אל סמוטפור (שיודע שהוא לא יישאר להרבה זמן, ויבוא עוד שר בקרוב אחריו) החליט על סדר מיון משלו, שעובד רק על פי מצב הרובריק (תחילה המטוסים, אחריהם כל הרכבים, ולבסוף כל מה שנשאר)

הסבירו מתי תשתמשו ב-comprator ומתי ב-comparable למיון הרשימה ומדוע. השלימו את הפונקציות sortByValue הבאה הממיינת רובוריקים לפי הנוסחה של פרופ' ריק, ואת פונקצית sortByMode עבור שר התחבורה.

```
public static void sortByValue(LinkedList<RoboRick> roboRicks){
        //**your implementation**
}
public static void sortByMode(LinkedList<RoboRick> roboRicks){
        //**your implementation**
}
```

Object Oriented Programing Exam Cheatsheet (2020)

- The APIs included in this sheet are incomplete - they only represent a small subset of the methods in the classes they refer to
- The provided methods shouldn't necessarily be used in order to fully answer the exam

Collections

class HashMap<K,V>	
V get(Object key)	Returns the value to which the specified key is mapped, or null if this map contains no mapping for the key
Set<K> keySet()	Returns a Set view of the keys contained in this map
V put(K key, V value)	Associates the specified value with the specified key in this map. If the map previously contained a mapping for the key, the old value is replaced.
V remove(Object key)	Removes the mapping for the specified key from this map if present
Collection<V> values()	Returns a Collection of the values contained in this map

class HashSet<E>	
boolean add(E e)	Adds the specified element to this set if it is not already present
boolean contains(Object o)	Returns true if this set contains the specified element
boolean remove(Object o)	Removes the specified element from this set if it is present
Iterator<E> iterator()	Returns an iterator over the elements in this set

class LinkedList<E>	
void addFirst(E e)	Inserts the specified element at the beginning of this list
void addLast(E e)	Appends the specified element to the end of this list
E getFirst()	Returns the first element in this list
E getLast()	Returns the last element in this list
E pollFirst()	Retrieves and removes the first element of this list, or returns null if this list is empty
E pollLast()	Retrieves and removes the last element of this list, or returns null if this list is empty
E get(int index)	Returns the element at the specified position in this list
int indexOf(Object o)	Returns the index of the first occurrence of the specified element in this list, or -1 if this list does not contain the element

class TreeSet<E>	
TreeSet()	Constructs a new, empty tree set, sorted according to the natural ordering of its elements
TreeSet(Comparator<? super E> comparator)	Constructs a new, empty tree set, sorted according to the specified comparator
boolean add(E e)	Adds the specified element to this set
Iterator<E> iterator()	Returns an iterator over the elements in this set in ascending order
boolean remove(Object o)	Removes the specified element from this set if it is present

Regex

class Pattern	
static Pattern compile(String regex)	Compiles the given regular expression into a pattern
Matcher matcher(String input)	Creates a matcher that will match the given input against this pattern

class Matcher	
boolean matches()	Attempts to match the entire region against the pattern
boolean find()	Attempts to find the next subsequence of the input sequence that matches the pattern
int start()	Returns the start index of the previous match
int end()	Returns the offset after the last character matched
String group(int group)	Returns the input subsequence captured by the given group during the previous match operation

Lambda Expressions

A general lambda expression structure:

```
(ARGUMENT_LIST) -> {BODY}
```

Examples:

```
(int x, int y) -> {int z = x+y; return z+1;}
(x,y) -> x+y
x -> foo(x)
(Cat c1, Cat c2) -> c1.compareTo(c2)
```