מדריך ל Parsing של השלבים בתרגיל 6

נכתב ע"י דוד סמואלסון
עבור הקורס "מבוא לתכנות מונחה עצמים" באוניברסיטת בר אילן
שמעביר ד"ר יואב גולדברג
פורסם ב 9/6/2018

DISCLAIMER

אל תסתמכו על הקוד שיש פה יותר מדיי. את כל הקוד שכתבתי פה כתבתי בnotepad במטרה להמחיש רעיון של איך אני ממליץ לעשות את הדברים, ולא במטרה לספק קוד שאמור לעבוד.
קחו את המדריך הזה בתור הכוונה כללית ולא בתור מדריך שמסביר לעשות משהו שלב אחרי שלב.
דבר נוסף – האחריות להימנע מחשד להעתקה היא עליכם. המלצה כללית – אל תחשבו על זה יותר מדיי, סה"כ לגיטימי להגיד שהיה את המדריך הזה ועקבתם אחריו, אבל אל תעתיקו מילה במילה.

יאללה שנתחיל?

דבר ראשון – לפני שנתחיל, תקראו את כל המשימה, ואת הפורמט של הקבצים, ותבינו איך זה עובד, איך שינוי בקובץ אמור לשנות את השלב, ואיך כל ה properties שם עובדים.

אחרי שעשינו את זה – נעבור לבלוק.

שינוי הבלוק:

החלק הזה דורש שבלוק יוכל לתמוך ברקעים שונים עבור hitPoints שונים, כאשר כל רקע יכול להיות או תמונה או צבע.
כנל לגבי outline – זה גם משתנה, לא כל בלוק בהכרח חייב לצייר מסגרת, וגם כשכן מציירים הצבע משתנה.

אז איך מממשים את זה?

קודם כל נתחיל מהרקע. המלצה שלי – נעשה interface בשם background, ששם תהיה פונקציה אחת שמקבלת rectangle ומציירת בתוכו את הרקע (תקראו לה איך שבא לכם, draw, drawAt וכו).
מה זה נותן לנו? אבסטרקציה של הרקע. רקע יכול להיות ממומש בכל מיני דרכים – בין אם רקע שמצייר תמונה, רקע שמצייר צבע, רקע שמכין לך קפה.. טוב נו נסחפתי.

משהו כמו זה:

```
public interface Background{

        void draw(Drawsurface d, Recrangle r);

}
```

את האינטרפייס הזה יממשו 2 מחלקות, מחלקה לכל סוג רקע שיש לנו – ImageBackground, שתהיה אחראית להציג תמונה בתור רקע, ו- ColorBackground, שכמובן אחראי להציג צבע בתור רקע.

עכשיו – איך נממש את השימוש שלהם בבלוק?

נתחיל מזה – בתור עצה להמשך, שתוכל לעזור בחלק הבא (לא יכול להסביר כרגע כי אנחנו עוד לא שם, אם אתם מתעקשים אז יצירת הבלוק בהינתן התיאור שלו), תעשו קונסטרקטור מינימלי שמקבל רק x,y (מבחינתכם height ו- width יהיו 0), ובשביל לשנות את שאר התכונות של הבלוק תוסיפו לכל תכונה פונקציית set.

בהמשך אוסיף קטע קוד שמראה איך זה אמור להראות.

חזרה לרקע :)
במקום צבע - בלוק יקבל Background, שישמש בתור הרקע הדיפולטיבי של הבלוק.

בשביל לתמוך ברקעים שונים עבור hitpoints שונים – נוסיף מפה שממפה ממספר (hitPoints עדכני) לרקע שנרצה לצייר עבור אותו hitPoint.

למה מפה – ולא רשימה? כי רשימה לא מספיק גמיש. אם נרצה למשל רקע מסויים כשלבלוק יש 4 נקודות, רקע רגיל ב 2 ו-3, ורקע מיוחד ב 1 – מפה מאפשרת לנו את הגמישות הזו, רשימה מוגבלת לסדר ספציפי.

אז מה עושים ב drawOn?
פשוט – אם קיים ערך במפה עבור ה hitPoint העדכני שלנו – נצייר אותו.
אחרת – נצייר את הרקע הדיפולטיבי (אם הוא קיים).
אם גם הרקע הדיפולטיבי הוא null, אז או שלא תציירו בכלל או זרקו אקספשן או ציירו סתם איזה איקס אדום על הבלוק.

ועכשיו – Stroke (כלומר מסגרת).

בדומה ל Background, נעשה קלאס של מסגרת שנקרא לו Stroke או Outline או איך שבא לכם, שם יהיה אותה פונקציה כמו ב Background שהיא draw(Drawsurface d, Rectangle r), נשמור אותו בתור member ב Block, ואם הוא לא null אז נצייר אותו.

סה"כ הקוד אמור להראות דומה לזה (תעשו טובה, אל תעתיקו כמו תוכים – תשתמשו בזה בתור רפרנס אבל בחייאת – אם כבר אתם מעתיקים, לא מילה במילה)

```java
public Block{
    private Rectangle r;
    private Background bg;
    private Map<Integer, Background> hpBackground;
    private Stroke stroke;
    // ... more properties

    public Block(Rectangle r){
        this.r = r;
        this.bg = null;
        this.stroke = null;
        this.hpBackground = new HashMap<>();
    }
    public Block(double x, double y){
        this(new Rectangle(new Point(x,y),0,0));
    }
    public Block(double x, double y, double width, double height){
        this(new Rectangle(new Point(x,y), width, height));
    }

    // setters for all other properties than x,y:
    // setWidth, setHeight
    // setBackground, setStroke, etc..

    // adds a background relative to a specific hitPoints
    public void addBackground(int hitPoint, Background bg){
        hpBackground.put(hitPoint, bg);
    }

    public void drawOn(DrawSurface d){
        if(hpBackground.containsKey(getHitPoints())){
            hpBackground.get(getHitPoints()).draw(d, getCollisionRectangle());
        }else if(bg != null){
            bg.draw(d, getCollisionRectangle());
        }else{
            // bg is null, you can draw some default background
            // signifies no background set.
        }

        if(stroke != null){
            stroke.draw(d, getCollisionRectangle());
        }
    }
```

חזרה ל- Parsing

העקרון הכי חשוב שנשמור עליו – זה להפריד משימות גדולות לתת משימות קטנות יותר, כי לעשות הכל ביחד בבת אחת זה קשה ולא חכם במיוחד.

אני לא אכנס יותר מדיי לפרטים – אבל אסביר את הרעיון מאחורי חילוק המשימות

מתחילים מזה שאנחנו מקבלים Reader, שמקושר לקובץ, ומשם אנחנו צריכים לקרוא את קובץ השלבים.

1. קודם כל נקרא את כל השורות בקובץ לרשימה של String (רשימה של שורות).
בזמן שנקרא את הקובץ נתעלם משורות שהן הערות (לא נוסיף אותם ל List).
אם משהו לא עבד עם ה Reader אז ייזרק אקספשן שאותו נצטרך לתפוס.

הערה: עלולים להיות הרבה דברים לא נכונים בדרך, אז במקום להסתבך עם איזה ערך נחזיר אם המידע לא מיוצג כמו שצריך – פשוט נזרוק RuntimeException (כי עבורו לא צריך להגדיר שהפונקציה זורקת אקספשן).

2. עכשיו בהינתן שהצלחנו לקרוא את הקובץ ושמרנו רשימה של שורות שבהם יש מידע על כמה שלבים - נרצה להפריד בין שלבים, כלומר לפצל את ה List<String> לרשימה של שלבים בודדים, כלומר ל List<List<String>>. לכן – צריך לכתוב פונקציה שיודעת לעשות את זה – כלומר מקבלת List<String> ומפצלת לתתי רשימות לפי מתי שכתוב START_LEVEL ו- END_LEVEL.
זאת גם הזדמנות מצויינת לבדוק תקינות קלט – באופן כללי הייתי מממש את זה באמצעות איזשהו Boolean שבודק האם אנחנו בתוך שלב (אם קראנו את START_LEVEL) – אם אנחנו בתוך שלב ואנחנו רואים שוב START_LEVEL - זרקו אקספשן, אם רואים END_LEVEL כשאנחנו לא בתוך שלב גם אפשר לזרוק אקספשן. בקיצור זה הרעיון.

3. עכשיו כשיש לנו רשימה של תיאורי שלבים (כשכל שלב זה רשימה של שורות), נרצה פונקציה שמקבלת תיאור של שלב ומפרסרת אותו לשלב – כלומר מקבלת List<String> ומחזירה אובייקט מסוג LevelInformation. זאתי תכלס משימה כבדה בפני עצמה – והייתי מפצל אותה גם לתתי שלבים, ולכן מעכשיו נתמקד באיך לפרסר שלב יחיד.

דבר ראשון – נשתמש ב Factory pattern – ניצור factory של שלב, שזה תכלס אובייקט שמחזיק את כל המשתנים של השלב (בדיוק מה שיש ב LevelInformation) שבהתחלה מאותחלים ל null, ולאט לאט כשנתקדם על הקובץ נעדכן את כל המאפיינים של השלב (באמצעות set) – עד שלבסוף נקרא בפנים לפונקציה create שתבדוק האם אתחלנו את כל המשתנים (כרגיל, אם לא אתחלנו את כולם אז נזרוק אקספשן), ומחזירה אובייקט מסוג LevelInformation.

לתיאור של שלב יש כמה חלקים עיקריים (לפחות ככה אני רואה את זה)

- תיאור של הבלוקים (מתחיל ב START_BLOCKS ונגמר ב END_BLOCKS)
- תיאור של מאפייני בלוקים (row_height, block_start_x, block_start_y) וכמובן כתובת של קובץ שמתאר את הבלוקים (block description file)
- תיאור של מאפיינים סתמיים יותר שקשורים לשלב (שם השלב, רוחב פדל, מהירות פדל, מהירויות כדורים וכו)

כלומר – יש לנו עוד הזדמנות לפצל פה למשימות:

1. פונקציה שמקבלת רשימת שורות של שלב ומחזירה רשימת שורות שנמצאים בין ה START_BLOCKS ל- END_BLOCKS, כלומר מחלצת את ה layout של הבלוקים

2. פונקציה שמקבלת רשימת שורות של שלב, מקבלת את ה levelFactory מחפשת את כל ה properties , מפרסרת אותם ועושה set בתוך ה factory.
עצה – משהו שחשבתי עליו בחלק הזה. בשביל לעשות את זה בצורה נוחה אפשר ליצור אינטרפייס בשם PropertyParser, שיש בו פונקציה אחת – parse(String line), שמחזירה void (כי היא אמורה לעשות set ולא להחזיר ערך מסויים)
את ה PropertyParser נממש עבור כל סוג property שיש לנו, ואז אפשר ליצור מפה (כמו dictionary מפייתון) מString ל- PropertyParser, ואז מה שנעשה זה ככה:

```
// Init_map:

Map<String, PropertyParser> parserMap = new HashMap<>();

parserMap.put("paddle_speed", new PaddleSpeedParser(levelFactory);

// … all the rest of the properties and property parsers we want to deploy

For each line in level:

    For each key in parserMap:

        If line starts with key:

                parserMap.get(key).parse(line);
```

אם אני לא טועה זה נקרא Strategy Pattern, אבל אל תסתמכו על איך קוראים לזה – פשוט תבינו את הקונספט ותראו איך אתם יכולים להשתמש בזה ככה שתכתבו קוד יותר נוח ויותר "טוב" (לפחות ככה אני חושב 😊)

3. פונקציה שמחלצת את המשתנים הבאים: row_height, block_start_x ו- block_start_y ומחזירה אותם באיזה struct או מערך או משהו כזה (אפשר ומומלץ לכתוב עבור זה parser ולעשות set באיזה מחלקה שתכתבו)
4. פונקציה שמוצאת את הכתובת של קובץ ה Block Definitions (לא חובה פונקציה אבל זה רעיונית משימה שצריך לעשות), בונה מזה Reader, ומשתמשת בפונקציה הסטטית של המחלקה BlocksDefinitionReader שיואב אמר לממש שמחזירה BlocksFromSymbolFactory.
תכלס – גם את זה צריך לפצל לפונקציות. צריך קודם כל לקרוא את כל השורות, להתעלם משורות ריקות ומהערות, ואחרי זה לעבור על הdata כמה שצריך בשביל לפרסר אותו.

אחרי שעשינו את כל זה, נצטרך לשלב בין כל המידע של הבלוקים וליצור רשימת בלוקים – כלומר פונקציה שמקבלת את ה blockFromSymbolsFactory, block properties (שזה ה x,y וגובה שורה) ואת ה block layout ומחזירה רשימת בלוקים. לא כזה מסובך – מתחילים עם y התחלתי לפי ה block properties, ועוברים שורה שורה על ה layout, כל פעם שנסיים שורה נגדיל את ה y ב row_height שגם יש ב properties.
על כל שורה נעבור תו תו, ונאתחל x התחלתי לפי ה start_x שב properties.
באמצעות ה blockFromSymbolFactory נבדוק כל תו:

- האם התו הוא block symbol – אם כן ניצור בלוק במיקום x,y העדכני באמצעות ה BlockCreator מתוך ה factory של הבלוקים, ונוסיף לרשימה.
- אחרת, אם התו הוא spacer – אז פשוט נקדם את ה x לפי ה width של אותו spacer (גם באמצעות ה factory של הבלוקים)
- אחרת – נזרוק אקספשן, כי יש symbol של בלוק לא ידוע.

אחרי שיש לנו את רשימת הבלוקים, כל מה שנותר לעשות זה לעשות set לרשימת הבלוקים ב levelFactory, וזהו פחות או יותר רק לעשות create והשלב אמור להיות מוכן (אם הכל הלך בסדר, אם לא אז אמור להיזרק אקספשן).

המלצה למימוש ה BlockCreator – מסובך ולא חובה אבל מלמד הרבה.

יואב פחות או יותר הכוון איך תעשו את זה – יש לכם BlockCreator, שמתאר factory של בלוק מסויים.
אז תאורטית, נוכל לעשות לממש את הממשק הזה, לשמור מלא משתנים שקשורים לבלוק, לעשות מלא set ואז ליצור.
אבל תכלס זה לא כזה יפה, ויש פתרון הרבה יותר נחמד. מסובך, אבל "טוב" יותר.

היו אמורים ללמד אתכם משהו שנקרא Decorator – שזה Design pattern שנותן מענה כשיש לנו מלא התנהגויות שאנחנו רוצים להוסיף לאובייקט מסויים בצורה שונה.

אז פתרון מגניב יהיה לעשות decorator על ה 'factory'ואז כל חלק ב decorator יעשה override ל-create, ויעשה set ל property שהוא אחראי אליו.

במימוש כזה יהיה לנו decorator לכל תכונה שיש בבלוק – רוחב, גובה, hitpoints, background, hitPointBackgroudn וכו.
מימוש של אחד כזה יהיה בערך ככה:

```
public class HitPointsBackgroundDecorator extends AbstractCreatorDecorator{
    private int hitPoint;
    private Background bg;
    public HitPointsBackgroundDecorator(int hitPoint, Background bg, BlockCreator decorated){
        super(decorated);
        this.hitPoint = hitPoint;
        this.bg = bg;
    }

    @override
    public Block create(int xpos, int ypos){
        Block b = super.create(xpos,ypos);
        b.addBackground(hitPoint, bg); // add background for specific hitpoint
        return b;
    }
}
```

נממש ככה את כל ה decorators , בכולם נעשה override ל create , שם ניצור בלוק לפי ה decorated (פשוט לקרוא ל super.create, שאמור להיות ממומש במחלקה האבסטרקטית ולקרוא ל decorated.create), אחרי זה נעשה set ל property ונחזיר את הבלוק.

איך נשתמש ב decorators האלה – בזמן שנפרסר את ה :block definitions file

דבר ראשון - נצטרך BlockCreatorFactory (יותר בכיוון של פונקציה סטטית כזאת), שבהינתן String שמתאר property של בלוק ובהינתן BlockCreator שעליו נרצה לעשות decorate, נחזיר את הדקורטור המתאים. אמור להיראות ככה:

```java
public static BlockCreator getDecorator(String property, BlockCreator decorated){
    String key,value;
    String[] split = property.split(":");
    if(split.length != 2) throw new RuntimeException("Unknown property '" + property + "'");
    key = split[0];
    value = split[1];
    if(key.equals("width"){
        return new WidthDecorator(Integer.parseInt(value), decorated);
    }else if(key.equals("height"){
        return new HeightDecorator(Integer.parseInt(value), decorated);
    }
    // else if for all properties, fill, fill-k, stroke etc...
    else{
        // unknown property
        throw new RuntimeException("Unknown property '" + property + "'");
    }
}
```

הקובץ לרוב מתחיל בשורות שמתארות ערכים דיפולטיבים – כרגע אתעלם מזה בשביל להסביר על יצירת בלוקים, ואחרי זה נתמודד עם ערכים דיפולטיבים.

בזמן שנעבור על הקובץ block definitiosn נסווג למקרים – האם זה bdef או sdef (מתעלמים מ default), ואם זה לא אף אחד מהם כרגיל נזרוק אקספשן.

אז מה זה בעצם לפרסר שורת bdef – זה בהינתן השורה ליצור BlockCreator מתאים (ליצור creator בסיסי ולהלביש עליו decorators).
צריך גם למצוא את ה symbol, ולהוסיף את הערכים האלה למפה (מפה מ String ל- BlockCreator),
כלומר נצטרך לשלוח את המפה לפונקציה שמפרסרת את שורות ה bdef ונוסיף את הcreator שיצרנו בפנים עם ה symbol שמצאנו בתור key.

איך נתמודד עם כל הערכים הדיפולטיבים? – ניצור decorator דיפולטיבי לפיהם.
כלומר – ניצור creator בסיסי (הרגיל הכי בסיסי, שפשוט יוצר בלוק במיקום x,y , בלי רוחב ואורך),
ועליו נתחיל להלביש decorators לפי אותם properties שיש ב default.
ובהמשך – במקום ליצור creator בסיסי בפונקציה שעושה parse ל bdef – נשלח את אותו creator דיפולטיבי לפונקציה והיא תשתמש בו בתור ה creator שאותו היא עוטפת עם decorators.

אם לא הבנתם מילה ממה שאמרתי – זה בסדר, או שתקראו על זה קצת באינטרנט ותקראו את התרגול על decorator אצלכם שוב או שתממשו את ה block factory בצורה ישירה, שיעבוד 😊