דגמי-עיצוב

דגמי עיצוב - design patterns - הם פתרונות מקובלים לבעיות שחוזרות על עצמן רבות במערכות-תוכנה. לכל דגם-עיצוב יש שם קצר וקליט המוכר על-ידי מתכנתים בכל העולם; הדבר מקל על תקשורת בין מפתחי-תוכנה.

הדגמים מחולקים לשלוש משפחות:

- דגמי יצירה (Creational patterns) - מתארים דרכים שונות ליצור עצמים חדשים.
- דגמי מבנה (Structural patterns) - מתארים דרכים שונות להרכיב עצמים פשוטים וליצור מהם עצמים מורכבים.
- דגמי התנהגות (Behavioral patterns) - מתארים איך עצמים מתקשרים ביניהם ומחלקים אחריות על משימות.

בכל משפחה יש הרבה דגמים; אנחנו נלמד כמה דוגמאות.

דגמי יצירה

בג'אבה יש דרך סטנדרטית ליצור עצמים חדשים - שימוש ב-new ובבנאי. אבל הדרך הזאת מאד מוגבלת: היא מייצרת עצם חדש בכל פעם - גם כשכבר קיים עצם זהה; והיא מחייבת אותנו לדעת את הסוג המדוייק של העצם שאנחנו רוצים ליצור.

כדי להתמודד עם המגבלה הזאת, נוצרו מספר דגמי-יצירה. לכל דגם-יצירה יש דוגמת-קוד; ניתן למצוא אותה בחבילה עם השם המתאים.

סינגלטון - Singleton

לפעמים אנחנו רוצים שיהיה עצם אחד ממחלקה מסויימת במערכת. לדוגמה: אנחנו בונים מחלקה שמתרגמת מעברית לאנגלית. אין טעם להחזיק יותר ממחלקה אחת כזאת בתוכנית. איך נעביר את המסר הזה למתכנתים שיבואו אחרינו?

אנחנו יכולים לכתוב הערה: "מתכנתים יקרים, אנא אל תבנו יותר מעצם אחד מסוג זה". אבל לא בטוח שהם יקראו.

במקום זה, עדיף להפוך את המחלקה "מתרגם" לסינגלטון. איך עושים את זה? שלב ראשון: הופכים את הבנאי שלה לפרטי. שלב שני: יוצרים משתנה סטטי מהמחלקה. שלב שלישי: כמה אפשרויות: אפשר להפוך את המשתנה הסטטי לציבורי וסופי (public final) ולאתחל אותו אתחול סטטי. אבל זה יגביל אותנו בעתיד אם נרצה שהאיתחול של המשתנה יהיה תלוי בזמן. פתרון טוב יותר הוא להפוך את המשתנה הסטטי לפרטי, ולהוסיף מתודה סטטית בשם getInstance(). המתודה הזאת תיצור את המשתנה הסטטי אם הוא לא קיים, ותחזיר אותו אם הוא קיים.

הערה: הדגם סינגלטון שנוי במחלוקת בין מהנדסי תוכנה. רבים אומרים שהוא דגם בעייתי הגורם לתלויות בין חלקי התוכנה ופוגע בעיקרון של הפרדת-אחריות (כמו משתנה גלובלי). אף יש שרואים בסינגלטון "אנטי-דגם". למרות זאת, חשוב להכיר את המושג "סינגלטון", הוא חלק משפת המתכנתים.

מפעל - Factory

לפעמים אנחנו בונים מחלקה עם הרבה פרמטרים, אבל רוב הלקוחות של המחלקה ירצו להשתמש רק בצירופים מסויימים של הפרמטרים. למשל: אנחנו בונים מחלקה "עובד" עם מחלקות יורשות "מתכנת" ו"מנהל"; אפשר לאתחל כל אחד מהם עם שם, משכורת, בונוסים וכו'. אבל, בכוח אדם יודעים לאתחל עובד רק לפי הדרגה שלו בחברה, שהיא מספר שלם. אם ינסו להפעיל בעצמם את הבנאי של המחלקה עם כל הפרמטרים, הם יסתבכו.

פתרון אפשרי הוא, להגדיר מחלקה בשם "מפעל לעובדים" - "EmployeeFactory". תהיה לה מתודה סטטית newEmployee שתקבל דרגה ותחזיר עובד מאותחל עם כל הפרמטרים המתאימים.

מפעל מופשט - Abstract factory

לפעמים אנחנו רוצים לבנות היררכיה שלמה של עצמים מסוג מסויים, אבל הסוג ידוע רק בזמן הריצה.

לדוגמה: אנחנו רוצים לייצר אוסף של עובדים, אבל, האיתחול של העובדים תלוי בסוג החברה שאנחנו נמצאים בה - קפיטליסטית, סוציאליסטית או קומוניסטית.

פתרון אפשרי הוא, לייצר ממשק של מפעל, ובו מתודה מופשטת (לא סטטית) newEmployee.

למפעל הזה יהיו מספר מימושים: CapitalistFactory, SocialistFactory, CommunistFactory.

בכל אחד מהם, המתודה newEmployee תמומש לפי המדיניות המתאימה.

אב טיפוס - Prototype

לפעמים אנחנו רוצים לבנות עצם שהוא העתק של עצם אחר נתון, עם כמה שינויים. לדוגמה: בתוכנית ציור, אנחנו רוצים לשכפל צורה מסויימת 20 פעמים בגדלים שונים, אבל הצורה והצבע צריכים להיות זהים.

הפתרון: להוסיף לעצם שלנו מתודת שיכפול - clone.

אגב, בג'אבה יש מתודה בשם clone במחלקה Object, אבל היא מוגנת (protected), קשה ולא מומלץ להשתמש בה. עדיף פשוט להגדיר את המתודה הזאת בעצמכם, אם אתם צריכים אותה.

דגמי מבנה

"משקל זבוב" - Flyweight

לפעמים ישנם עצמים שהזהות שלהם תלויה רק בפרמטרי האיתחול. לדוגמה: נקודה במרחב. אנחנו רוצים למנוע יצירת עצמים מיותרים.

הפתרון: להוסיף למחלקה משתנה סטטי פרטי מסוג Map. להפוך את הבנאי לפרטי. להוסיף מתודה סטטית newInstance. המתודה תבדוק אם המשתנה כבר קיים במפה. אם כן - היא תחזיר אותו; אם לא - היא תיצור חדש ותכניס למפה.

הדגם הזה כבר נמצא בשימוש בג'אבה, במחרוזות. שתי מחרוזות קבועות עם אותן אותיות, הן אותו עצם. (אם יוצרים מחרוזת מחיבור של שני משתני-מחרוזת, אז התוצאה יכולה להיות שונה. אבל אפשר לקרוא למתודה intern כדי לקבל קישור לעצם הייחודי המתאים לתוכן המחרוזת. ראו דוגמה בקובץ StringInternDemo).

הרכב - Composite

יש עצמים רבים שהמבנה הלוגי שלהם הוא עץ. למשל: מסמך xml, תיקיה במחשב, אוסף עצמים גרפיים.

בעץ יש צומת פנימי ועלה, ואנחנו רוצים להתייחס לשניהם באותה צורה בלי משפט if.

הפתרון הוא להגדיר מחלקה מופשטת המציינת צומת כלשהו בעץ; למחלקה זו יהיו שתי מחלקות יורשות - צומת פנימי וצומת סופי.

מתאם - Adapter

לפעמים יש לנו מחלקה שעושה את מה שאנחנו רוצים, אבל לא בממשק שאנחנו רוצים.

לדוגמה: יש לנו Iterator, אבל אנחנו צריכים Iterable כדי לעבור עליו בלולאת for של ג'אבה.

הפתרון: כמו שבבית משתמשים במתאם בין תקע של אינטרנט לשקע של טלפון, כך גם בתוכנה בונים מחלקה שמתאמת בין המחלקה שיש לנו, לבין הממשק שאנחנו רוצים לממש.

בא-כוח - Proxy

לפעמים יש לנו מחלקה שיש בה מתודות "מסוכנות". למשל, מחלקה המאפשרת לבצע פעולות כלשהן של מערכת ההפעלה.

אנחנו רוצים לתת ללקוחות שלנו לבצע פעולות, אבל רק אחרי שבדקנו וראינו שהפעולה לא מסוכנת.

הפתרון: לעטוף את המחלקה המסוכנת במחלקה "פרוקסי" שמכילה רק את הפעולות הלא מסוכנות.

דגמי התנהגות

מתודת תבנית - Template method

לפעמים אנחנו צריכים לממש אלגוריתם מסויים בכמה דרכים שונות, הנבדלות בפרטים, אבל דומות במבנה הכללי.

לדוגמה, אנחנו רוצים לבצע הדמיה ומדידת זמן של חיפוש 10 עצמים במערך. אנחנו רוצים לבדוק שני אלגוריתמים: (א) קודם לסדר ואז לבצע חיפוש בינארי, (ב) לא לסדר, רק לבצע חיפוש ליניארי. בשני המקרים, הבדיקה היא די דומה - זוכרים את הזמן בהתחלה, מסדרים (אם צריך), מחפשים, מודדים את הזמן בסוף, ומחשבים את ההפרש.

אפשר לחסוך את הקוד המיותר ע"י כתיבת מחלקה מופשטת SearchSimulator שבה יהיה האלגוריתם "בראשי פרקים", עם מתודות מופשטות עבור הפרטים - מתודה לסידור ומתודה לחיפוש.

למחלקה הזאת יהיו שתי מחלקות יורשות - אחת לסידור וחיפוש בינארי, והשניה לחיפוש ליניארי. כל אחת מהמחלקות הללו תממש את הפרטים בצורה שונה: בראשונה הסידור ישתמש ב-sort של ג'אבה

(או בשיטה אחרת לסידור) והחיפוש ישתמש ב-binarySearch של ג'אבה; בשניה הסידור יהיה ריק והחיפוש ישתמש בחיפוש ליניארי (המתודה contains).

צופה - Observer

לפעמים יש עצם שהמצב שלו משתנה, ויש הרבה עצמים שרוצים להגיב לשינוי הזה. לדוגמה, חישבו על אפליקציית צ'ט. בכל פעם שיש הודעה חדשה בחדר, אנחנו רוצים שכל תוכנות-הלקוח המחוברות לחדר הזה יראו את ההודעה החדשה. אפשר להגיד לכל תוכנות-הלקוח לבדוק כל הזמן האם המצב בחדר השתנה, אבל זה די בזבזני. במקום זה, עדיף להעביר את האחריות לחדר. החדר יודע מתי הוא משתנה - וכשהוא משתנה, הוא מודיע לכל ה"צופים".

לשם כך, החדר (שנקרא גם Subject) צריך להכיל מתודה שמאפשרת לצופים להירשם: register. הוא שומר את כל הצופים שנרשמו אצלו ברשימה, וכשהמצב שלו משתנה, הוא מפעיל מתודת עדכון (update) אצל כל הצופים.

שיכתוב לפי דגמים - Refactoring

במקרים רבים תיתקלו בקוד שאינו כתוב טוב - יש בו כפילויות, טעויות וחוסר-בהירות. אחת הדרכים לסדר את הקוד היא לשכתב אותו בהתאם לדגמי-עיצוב. בתיקיה exercises תמצאו מספר תרגילים לשיכתוב לפי דגמים. בהצלחה!

מקורות

- Joshua Kerievsky, "Refactoring to Patterns Catalog", https://www.industriallogic.com/xp/refactoring/catalog.html
- מצגת של קרן כליף ממכללת אפקה.
- Dane Cameron, "A Software Engineer Learns Java and OOP".

סיכם: אראל סגל-הלוי.