**designPaterrns**

**Singletone**

תבנית עיצוב יצרנית המבטיחה שלמחלקה יהיה רק מופע אחד, תוך מתן נקודת גישה גלובלית למופע זה.

**מימוש:**

1. משתנה סטטי פרטי למחלקה שמאחסן את מופע המחלקה
2. מתודה “GetInstance” סטטית ציבורית שמחזירה את המופע שנוצר ורק במידה והוא עדיין לא קיים (null) אז תקרא קודם לבנאי שייצור אותו.
3. ctor סטטי פרטי, כדי שלא יוכלו ליצור / להשתמש בו מבחוץ, אבל ירוץ בהתייחסות ראשונה למחלקה.

**Factory** **Method**

תבנית עיצוב יצרנית, שעוקפת את השימוש הישיר באופרטור New ע"י כימוס תהליך יצירת האובייקטים (שיורשים כולם מאותו אבא), ע"י קריאה לפונקציה שאחראית על בניית האובייקט הנדרש.

**צורת המימוש המורחבת:**

1. האובייקטים ליצירה:

* ממשק/מחלקה אבסטרקטית שכולם ירשו ממנה, עם הפונקציה Operation.
* מחלקות ספציפיות שיורשות מהאבא וממשות את הפונקציה באופן ספציפי.

1. המפעלים:

* מחלקת אבא Factory כללית, עם מופע יחיד, פונקציה וירטואליתFactoryMethod שמחזירה אובייקט, פונקציהCreateNew שיוצרת אובייקט חדש לפי תנאי – מאתחלת את המופע להיות מפעל ספציפי, ופונקציה SomeOperation שבה נקראתFactoryMethod ומופעלת הפונקציה Operation על האובייקט שחזר מהפונקציה.
* במחלקותFactory הספציפיות שיורשות מהכללית יש מופע יחיד, ו FactoryMethodשמחזירה אובייקט ספציפי.

1. בתוכנית הראשית: יוצרים Factory. ואז: factory.CreateNew(condition).SomeOperation()

**Abstract Factory**

תבנית עיצוב יצרנית, שמאפשרת יצירת משפחות אובייקטים קשורים, לא דווקא כאלה שיורשים מאותו אב, ע"י יצירת מפעל מתמחה ספציפי בכל פעם ועקיפת האופרטור New.

מימוש:

1. האובייקטים:

* ממשק לכל סוג אובייקט מתואר (ob1, ob2).
* מחלקות ספציפיות שיורשות את הממשקים ונותנות להם תיאור ייחודי (type1, type2).

1. המפעל:
   * + ממשק מפעל אבסטרקטי שיש לו פונקציות יצירת אובייקטים (CreateOb1,CreateOb2)
     + מפעלים ספציפיים שממשים את הממשק ומחזירים אובייקטים מהתיאור שלהם (type1, type2).
2. בתוכנית הראשית: יוצרים מפעל ספציפי, ועליו מפעילים את הCreatים.

ניתן לכתוב פונקציות שיבדקו שהאובייקטים המיוצרים תואמים זל"ז.

(לדוג': אם יש לי ממשקים של רוגלאך ובורקס, ומפעלים של תנור בשרי ותנור חלבי, כשאני מבקשת סתם בורקס מתנור חלבי, אני יודעת שזה בורקס חלבי...)

**Builder**

תבנית עיצוב יצרנית, המאפשרת לבנות אובייקטים מורכבים (או עם פרמטרים אופציונליים) באופן מודולרי, צעד אחר צעד.

במקום לחייב לשלוח הכל דרך הctor, או להסתבך עם שליחה של שדות מורכבים שהם בעצמם אובייקטים, יוצרים "בונים" וקוראים רק לאלו מתוכם שאנו מעוניינים בהם.

מימוש:

* ממשק שיש לו פונקציות שלבי עבודה (במספר השלבים האופציונליים)
* יורשים שיש להם מופע פרטי של האובייקט הנבנה וממשים את שלבי העבודה וקוראים בכל פעם לפונקציות הרצויות במחלקת האובייקט.

בנוסף יש להם פונקציה Reset שמאתחלת את האובייקט לחדש, ופונקציה GetProduct שמחזירה את האובייקט הגמור ואז קוראת לאתחול.

(כל יורש יכול לבנות אוביקט מסוג מחלקה אחרת)

ניתן ליצור "מנהלי עבודה" שהם יקראו לבונים מסויימים וייצרו אובייקטים בשלבים מתוכננים מראש:

* מופע של Ibuilder עםset ציבורי וget פרטי.
* פונקציות minimalLevels, maxLevels וכדו' שקוראות לפונקציות של שלבי הבונה.

בתוכנית הראשית:

ניתן ליצור בונה ספציפי לאובייקט שרוצים לייצר, ולקרוא לפונקציות של השלבים הרצויים, ולהחזיר את האובייקט.

לחילופין ניתן ליצור מנהל, לאתחל את הבונה שלו בבונה הספציפי, ואז לייצר בבת אחת את כל השלבים מהפונקציות שהגדרנו.

**ProtoType**

תבנית עיצוב יצרנית, שמאפשרת ליצור בתוכנית הראשית העתק מאובייקט, בלי לדעת את המימוש הפנימי הפרטי שלו ובלי להסתבך עם refernce...

יוצרים ממשק IClone ובו פונקציה Clone. במימוש הפונקציה במחלקה: נייצר אובייקט חדש מסוג המחלקה, נעתיק אליו בהעתקה רדודה את הvalueType ובהעתקה עמוקה את ה refernceType, ואז נחזיר.

בתוכנית הראשית פשוט נקרא לפונקציה: Object o2 = o1.Clone().

**Composite**

תבנית עיצוב מבנית, שמאפשרת להתייחס בצורה אחידה לאובייקטים מורכבים ולאובייקטים פשוטים, המממשים את אותו ממשק, ומחוברים למבנה עץ.

מימוש:

יוצרים מחלקה אבסטרקטית כללית, שבה יש פונקציות Add,Remove,IsComposite, והפונקציה Operation שהיא תמומש באופן שונה ע"י הרכיבים השונים. (ובנוסף members שיהיו לכל היורשים).

מהמחלקה ירשו לפחות שתי מחלקות: רכיב פשוט ורכיב מורכב.

* ברכיב פשוט: Add,Remove יחזירו שגיאה, IsComposite יחזיר false, Operation תעשה פעולה על הרכיב.
* ברכיב מורכב: יהיה גם משתנה של List של אובייקטים מסוג המחלקה האבסטרקטית, Add ו-Remove ינהלו את האבירים ברשימה הזו. IsComposite יחזיר true. Operation תרוץ על איברי הרשימה, ותפעיל על כל אחד מהם את Operation שלו. (בקיצור: רקורסיה)

**Adapter**

תבנית עיצוב מבנית, המאפשרת למחלקת Client המצפה לקבל ממשק מסוים, להשתמש באובייקט Service שיש לו ממשק אחר. או: מגדירה מחלקה חדשה המתנהגת בדומה למחלקה קיימת, עם נתונים נוספים.

מימוש: מחלקת "Adapter", ובה שדה של מחלקת "Service", ושיטות המשתמשות ביכולות של המופע, ומבצעות התאמה לממשק המצופה.

**Observer**

תבנית עיצוב התנהגותית, המאפשרת תקשורת בין אובייקטים שאינם קשורים זל"ז בירושה, ע"י מנגנון הרשמה.

מימוש:

* הנושא(/מפרסם) הוא האובייקט המחזיק ברשימת האובייקטים המשקיפים עליו ולוקח אחריות על עדכון הצופים בעת שינוי. יש לו פונקציות: Attach, Detachשאחראיות על הוספת והסרת משקיפים לרשימה, ופונקציה Notify שרצה על רשימת המשקיפים ומפעילה עליהם את Update.
* המשקיף(/מנוי) הוא מי שנרשם לנושא ויקבל התראה/הודעה/הפעלת פונקציה. כל המשקיפים מממשים את אותו ממשק (יש להם פונקציה Update שתופעל כשיקרה האירוע, היא יכולה לקבל כפרמטר את הנושא או את מהות השינוי).

(לדוג': הרבה בעלי חיים שמחכים לשינוי במזג האוויר. מזגאוויר זה הנושא, בע"ח הם המשקיפים, ותופעל להם הפונקציה sound() כשיופעל set של מזג האויר)

**Template**

תבנית עיצוב התנהגותית, המחלקת תהליך לשלבים שונים, על מנת להקל על יצירת תהליכים **דומים** עם פרטי מימוש **שונים**, ומאפשרת התאמה אישית של התהליכים.

מימוש:

* יצירת מחלקת בסיס מופשטת המכילה את השלבים השונים של התהליך. השלבים יכולים להיות מוגדרים כאבסטרקטיים, כך שהמחלקות היורשות יהיו חייבות לממש אותם. חלק מהשלבים יכולים להיות מוגדרים עם מימוש ברירת מחדל, שניתן לשנות על ידי המחלקות היורשות.

שימוש בשיטת Template() המוגדרת כ-Final, שקוראת לשיטות לפי סדרן בתהליך.

* המחלקות היורשות חייבות לממש את כל השלבים המופשטים, ויכולות לתת מימוש יחודי לשלבי ברירת המחדל.
* ניתן להרחיב עם Hook – צעדים אופציונלים עם גוף ריק, שאין חובה לממש.