Design Patterns

הקוד עבור שלושת הדוגמאות, נמצא בלינק הבא
https://github.com/dani1996dani/Design-Patterns

Adapter Pattern (Structural Pattern)

ה - adapter pattern בא לגשר בין שני אינטרפייסים או מחלקות אשר קיימת ביניהם חוסר תאימות. לדוגמא, יש לחשוב על מקרה בו לי יש אייפון, ואין לי סוללה. שכחתי את המטען שלי בבית, והמטען היחיד שזמין לי כרגע הוא מטען של גלקסי.
כל מה שיש עליי לעשות הוא להשתמש במתאם (adapter). אני אשתמש ביכולת הטעינה של מטען הגלקסי על מנת להטעין את האייפון שלי.
במחלקה ChargerWork יש אובייקט מסוג Charger (שהוא אינטרפייס) אשר מוכן לקבל כל מטען אשר מממש את האינטרפייס הזה. אך שימו לב, אשר המחלקה GalaxyCharger **לא ממשת** את האינטרפייס הזה, ובנוסף מתודת הטעינה שלה נקראת בשם אחר. לכן, עלינו ליצור אובייקט מסוג ChargerAdapter אשר **כן** ממשת את האינטרפייס (כלומר, תהיה לו את המתודה עם בשם אשר ChargerWork מכיר ויכול להפעיל). בתוך ChargerAdapter ניצור אובייקט מסוג GalaxyCharger, ובתוך מתודת הטעינה של ChargerAdapter נקרא למתודת הטעינה של GalaxyCharger. כך, נוכל להעביר את ChargerAdapter כאובייקט מסוג Charger, נוכל להפעיל את מתודת הטעינה שלו, ולהטעין את האייפון שלי.

```
"C:\Program Files\Java\jdk10\bin\java.exe" "-javaagent:E:\Inte
Thank god I found a charger, I am at 50%
I Think I'm Charging a Galaxy Phone Now, Currently at: 55%
I Think I'm Charging a Galaxy Phone Now, Currently at: 60%
I Think I'm Charging a Galaxy Phone Now, Currently at: 65%
I Think I'm Charging a Galaxy Phone Now, Currently at: 70%
I Think I'm Charging a Galaxy Phone Now, Currently at: 75%
I Think I'm Charging a Galaxy Phone Now, Currently at: 80%
I Think I'm Charging a Galaxy Phone Now, Currently at: 85%
I Think I'm Charging a Galaxy Phone Now, Currently at: 90%
I Think I'm Charging a Galaxy Phone Now, Currently at: 95%
I Think I'm Charging a Galaxy Phone Now, Currently at: 100%

Process finished with exit code 0
```

Prototype Pattern (Creational Pattern)

ה - prototype pattern נועד לחסוך בפעולות מסובכות על מנת ליצור אובייקטים חדשים.
בדוגמא שהרכבתי, יש שרת אשר יוצר עבורנו אובייקטים של צורות גיאומטריות.
כידוע, תקשורת בין שרת לקליינט היא פעולה יקרה (פעולת I/O). בנוסף, היא יכולה להיכשל, דבר שעלול לפגוע בקליינט. על מנת לחסוך בפעולות מסוג זה, יש להתחבר לשרת פעם אחת, על מנת לקבל את כל הצורות שאנו רוצים להשתמש בהם. לאחר מכן, אין צורך להתחבר שוב לשרת, מכיוון שיש בידינו כבר אובייקטים של הצורות (אפשר לחשוב על האובייקטים הראשונים כאב טיפוס, prototype). כל מה שנותר עלינו לעשות, הוא לשכפל את כל המידע השוכן באובייקט הקיים (אב הטיפוס), לאובייקט החדש.
את פעולה זו, נעשה באמצעות המתודה clone, אשר מקורה הוא במחלקה Object.
שימו לב שאי אפשר להשתמש במתודה הזאת על כל אובייקט, מכיוון שהמתודה clone היא protected. לכן, על מנת להשתמש בה, עלינו לשאול את עצמנו אילו אובייקטים אנו רוצים לשכפל. עבור כל מחלקה כזו, יש לממש את האינטרפייס Cloneable.
לאחר מכן יש לדרוס את המתודה clone בתוך המחלקה המממשת את האינטרפייס הזה. בתוך המתודה אשר אנו דורסים, ניצור אובייקט ריק חדש מהטיפוס הרצוי, נעתיק אליו את הערכים הרצויים מאובייקט האב טיפוס, ונחזיר את האובייקט בסוף המתודה.

```
"C:\Program Files\Java\jdk10\bin\java.exe" "-javaagent:E:\IntelliJ IDEA
Original Objects

Oval{id=64, type='Oval', area=21}
Square{id=65, type='Square', area=65}
Triangle{id=66, type='Triangle', area=94}

Cloned Objects

This is the clone oval Oval{id=64, type='Oval', area=21}
This is the clone square Square{id=65, type='Square', area=65}
This is the clone triangle Triangle{id=66, type='Triangle', area=94}

Process finished with exit code 0
```

Visitor Pattern (Behavioural Pattern)

ה - visitor pattern נוצר בשביל שנוכל לבצע פעולות שונות (להפעיל אלגוריתמים שונים) על אובייקטים מטיפוסים שונים (אשר מאפשרים ביצוע פעולות כאלו).
בדוגמא, אנו רוצים לחשב מס עבור מוצרים מטיפוסים שונים. יש בדוגמא שלושה טיפוסים שונים, מוצר יסוד (Necessity), מכונית ובית.
בחישוב מס רגיל (BasicTaxVisitor), לא נגבה מס על מוצר יסוד, על מכונית נגבה מס של 20%, ועל בית נגבה מס של 25%.
לעיתים, נרצה להפעיל מס גבוה יותר, לכן נשתמש בשיטה שונה לחישוב המס (באלגוריתם שונה). במחלקה GreedyTaxVisitor, זה בדיוק מה שקורה - על מוצר יסוד נוסיף 2 שקלים של מס, רכישת מכונית תזכה אותנו במס 100% ורכישת בית, במס של 50%.
על מנת שנוכל לחשב מס על אובייקט מסוים, נממש במחלקה שלו (לדוגמא, בCar), את האינטרפייס Visitable. יש בו רק מתודה אחת, accept, אשר מקבלת כפרמטר Visitor (טיפוס זה הוא אינטרפייס, על מנת שנוכל לקבל visitors מטיפוסים שונים ומשונים, ולמנוע נעילה לטיפוס visitor יחיד).
באינטרפייס Visitor, יש מתודה בשם visit, אשר מקבלת פרמטר אחד, שהוא האובייקט עליו מחשבים את המס. מכיוון שאנו רוצים לחשב מס על סוגי אובייקטים שונים, נשתמש ב - method overloading, כאשר שם המתודה נשאר זהה, אך טיפוס הפרמטר משתנה. מספר המתודות יהיו כמספר הסוגים השונים שאנו רוצים לחשב עליהם מס.
במתודה accept, נפעיל את האלגוריתם של ה-visitor, אשר יחשב את הסכום בתוספת מס, יחזיר את הסכום למתודה accept, והמתודה עצמה תחזיר את הערך החדש שקיבלה.

```
"C:\Program Files\Java\jdk10\bin\java.e
Basic Tax - Milk Price: 3 ₪
Basic Tax - Honda Price: 60000 ₪
Basic Tax - Mansion Price: 25000000 ₪

Greedy Tax - Milk Price: 5 ₪
Greedy Tax - Honda Price: 100000 ₪
Greedy Tax - Mansion Price: 30000000 ₪

Process finished with exit code 0
```