**הרצאות OOP - פרק 2**

**2.1 עיצוב תוכנה – אנקפסולציה ו-API:**

**אנקפסולציה:**

* פונקציה – אוסף של פעולות בסיסיות שאחראיות על פעולה יותר מורכבת
* מבנים – אוסף של משתנים שקשורים לאיזשהו תחום אחריות.
  1. מבנים אפשר לשלוח מפונקציה אחת לאחרת
* לכל תחום אחריות יהיה לנו מבנה של כל המשתנים שקשורים אליו וכמה פונקציות שקשורות אליו
* עיקרון האנקפסולציה (כימוס) –
  1. לחלק את הקוד לקפסולות
  2. מורכבת משני חלקים:
     + לאחד ליחידה אחת את כל הפונקציות והמשתנים שעוסקים באותו תחום אחריות
       - למשל באיקס עיגול – יחידה שאחראית על השחקן, יחידה שאחראית על עיצוב, יחידה האחראית על הלוח
       - המושג עצם נולד כדי לקחת את הרעיון התיאורטי של האנקפסולציה ולגלם אותו בקוד – העצם הוא הישות שמאגדת את המשתנים והפעולות ששייכים לאיזשהו תחום אחריות.
     + הסתרת מידע - כל עצם מפריד בין מה שמשרת יחידות אחרות ובין מה שנועד לשימוש פנימי
       - חוץ מכמה פונקציות כל עצם מסתיר מאחרים את המשתנים והפונקציות שלו כאילו לא קיימים
  3. למעשה המשמעות של אנקפסולציה היא חלוקת כל הפונקציונליות שבתוכנה לקפסולות שמרכזות תחומי אחריות, והקפסולות מסתירות כמה שיותר דברים
     + את שדות האובייקט כמעט אף פעם לא נחשוף
* יתרונות האנקפסולציה:
  1. דיבוג קל יותר – הקוד מחולק לחלקים לוגים ומאפשר למקד בדיקות לאיתור תקלות
  2. רכיב שעומד בפני עצמו ניתן לשלב גם בתוכנות אחרות
  3. חלוקת עבודה בין חברי הצוות
  4. שינוי מימוש מחלקה אחת לא מצריך שינויים במחלקה אחרת

**מוטיבציה להסתרת מידע:**

* יתרונות:
  1. מאלצת אותנו לכתוב קוד שקשור לתחום אחריות של אובייקט בתוך האובייקט עצמו
  2. הסתרת מידע גורמת לאנקפסולציה
  3. פיתוח אינקרמנטלי – תכנון שינויים מראש – מכינה גרסה ראשונית לפרויקט, בודקת אותה, ומבצעת כל פעם שינוי באובייקט אחר
  4. קל יותר להתמצא
  5. פחות באגים
  6. קל לדבג

אנקפסולציה –

* עצם/אובייקט הינו אוסף משתנים ופונקציות הקשורים לאותו תחום אחריות
* אחת המטרות העיקריות של הכימוס היא הפרדה בין פונקציות ומשתנים הנועדו לשימוש

פנימי בתוך המחלקה לבין אלו הנועדו לשימוש חיצוני מחוץ למחלקה.

* עיקרון הכימוס מנחה אותנו להסתיר מידע על המימוש

**מגדירי נראות - Visibility Modifiers:**

* שתי מילים שמורות בג'אווה – public, private – מגדירות את הנראות של האיבר:
  1. איברים (שדות / שיטות) המסומנים כpublic זמינים מחוץ לעצם ואילו ניסיון גישה לאיברים המסומנים ב-private ייכשל בקומפילציה
  2. נכתוב אותן בתחילת שורת ההצהרה על האיבר
* כל מה שמסומן ב-public חשוף לגישה של עצמים אחרים,

וכל מה שמסומן כprivate ניתן לגשת אליו רק מתוך המחלקה (מוסתר)

* גם לפני קבועים צריך לציין public / private
  1. קבוע הוא מאפיין של כל המחלקה והגישה אליו היא דרך שם המחלקה
* שדות כמעט תמיד יהיו פרטיים
* גישה לשדות או מתודות פרטיות תיצור שגיאה כבר בזמן קומפילציה
* אנו מעדיפים שגיאות קומפילציה כי כך נוכל תקן את הבעיות לפני שהופכים לבאג שנצטרך לחפש בזמן הריצה
  1. בגלל זה אנו משתמשים בpublic/private, static, void וכדומה כדי לתת מידע לקומפיילר שידע לעזור לנו
* בגלל זה PYTHON מתאימה יותר לכתיבת תוכנית וJAVA מתאימה יותר לבניית תוכנה
* חשוב – שיטה יכולה לגשת לכל איבר במחלקה שלה, גם אם הוא פרטי והיא לא

**API – Application Programming Interface:**

* לאיברים הפומביים של מחלקה אנו קוראים הAPI של המחלקה
* הAPI הוא הממשק התכנותי של המחלקה – הממשק של העולם החיצון עם המחלקה
* האיברים הפרטיים לעומת זאת הם רק חלק מהמימוש
* כשאנו כותבים קוד מחוץ למחלקה שמשתמש במחלקה מעניין אותנו הAPI שלה – רק מה שאנו יכולים לגשת אליו
* ההפרדה בין פרטי לפומבי היא מאוד משמעותי
* הרבה פעמים בכתיבת המחלקה נפריד בין השדות הפרטיים והפומביים ובין השיטות הפרטיות והפומביות כדי לסדר לנו מה נכנס לAPI
* כשאנו כותבים קוד חדש נעבוד ב-3 שלבים:
  1. אנקפסולציה – חלוקת העצמים לקפסולות לפי תחומי אחריות
  2. קביעת ה-API לכל עצם
  3. מימוש המחלקה
     + יכול להיות שבשלב זה נרצה להוסיף קבועים, פונקציות או שדות – כל אלו יהיו פרטיים
* כל האיברים שנוסיף בשלב המימוש יהיו פרטיים

האיברים הפומביים יהיו האיברים המגדירים את החוזה – הAPI

* החתימה של השיטה main חייבת להיות מוגדרת כפומבית – אפשר לחשוב עליה בתור הממשק התכנותי של התכנית כולה כלפי מערכת ההפעלה – public static void main(String[] args)

**Minimal API:**

* אנקפסולציה אומרת שכל עצם נולד כדי למלא תפקיד מוגדר עבור עצמים אחרים. תוכנה מונית עצמים היא רשת לקוחות וספקי שירות.
* רוב העצמים משתמשים בשירות של עצמים אחרים וכל העצמים מספקים שירות (אולי חוץ מ-main).
* השירות בא לידי ביטוי ב-API
* צורת המחבה המרכזית בתכנות מונחה עצמים היא החלפת כובעים – כשנפתח מחלקה מסוימת נבחן אותה מנקודת המבט של הלקוח.
* בסופו של דבר הלקוח שלנו הוא בן אדם, הממשק צריך להיות נוח, אינטואיטיבי ופשוט.
* עיקרון הממשק המינימלי הוא עיקרון שאומר KEEP IT SIMPLE – האובייקט צריך לעשות כל מה שמצופה ממנו ורק מה שמצופה ממנו, כל המוסיף גורע.
  1. מנקודת המבט של הלקוח יותר פונקציונליות זה יותר להבין, יותר לזכור ויותר מקום לטעויות.
  2. מנקודת המבט של כותב הקוד
* דוגמא – איקס עיגול:
  1. נרצה שבלוח יהיו הפונקציונליות הבאות:
     + יצירת הלוח – בנאי Board()
     + לשים סימן על הלוח – putMark()
     + לקרוא את הסימן במקום מסוים בלוח – getMark(int row, int col)
  2. אבל ניתן להוסיף פונקציונליות כמו:
     + מחיקת הלוח עבור משחק חדש – בשביל זה אפשר ליצור פשוט מופע חדש
     + פונקציה שאומרת איפה כדאי לשים סימן חדש – אחריות נוספת ששוברת את האנקפסולציה של מחלקה אחרת – של השחקן
* יתרונות של API מינימלי – כל המוסיף גורע
* עדיף להוסיף מלהוריד שיטות שקוד אחר כבר תלוי בהם.

**אבסטרקציה:**

* אבסטרקציה היא התרחקות מהמימוש והתקרבות לצורך הלקוח.
* ממשק המשתמש עובר תהליך אבסטרקציה – הלקוח לא יודע איך המימוש מתבצע
* לאבסטרקציה יש יתרון לא רק מצד הלקוח אלא גם למפתחים – בתהליך האנקפסולציה קיבלנו את מה שהעצם צריך לעשות, וזה החלק הקבוע, החלק של איך הוא עושה זה החלק שנתון לשינוי.
* כל עוד הAPI אבסטרקטי הוא לא משתנה גם אם המימוש משתנה, ואם הAPI לא משתנה אז השינויים לא מתפשטים.
* שני עקרונות שעוזרים להנחות את ההגדרה של הAPI של מחלקה:
  1. ממשק מינימלי –
     + עוזר לספק פשטות
     + שמירה על אנקפסולציה – שומר שהמחלקה לא לוקחת תפקיד של מחלקה אחרת
  2. אבסטרקציה –
     + הפרדה בין מה המחלקה נועדה לספק לבין איך היא עושה את זה
     + נותנת ממשק נוח ללקוח, מגן עליו מפני שינויים במימוש וגם חוסך ממנו את הפרטים הקטנים.

סיכום שלבי הפיתוח של תכנית מונחית עצמים קטנה:

1. אנקפסולציה: חלק את התכנית למחלקות, כך שכל מחלקה אחראית על נושא אחד מוגדר היטב.

2. קביעת הAPI : אמרנו ש-API הוא הממשק התכנותי של המחלקה application programming interface. כאן אנחנו מבטאים את התפקיד של המחלקה בבחירת האיברים פומביים שלה. דיברנו גם על שני עקרונות שעוזרים לנו בשלב קביעת ה-API:

2.1 minimal API - לא להכביד על הממשק. להעדיף רשימת שיטות פשוטה להבנה ופיענוח.

2.2 אבסטרקציה: קודם לחשוב איך היינו רוצים שהמחלקה תיראה מבחוץ. איך ניתן שירות נח. השירות עשוי להיות (וכנראה גם כדאי שיהיה) שונה מאוד מאיך שחשבנו לממש את הפונקציונליות בפועל.

3. מימוש: כאן אנחנו מממשים את ה-API שהחלטנו עליו. בשלב הזה אנחנו יכולים להוסיף קבועים, או שדות, או שיטות - לפי הצורך. בתנאי --- שהאיברים הללו פרטיים - הם לא חלק מהממשק. לזה קראנו הסתרת מידע, וזהו חלק מהותי בשימור החלוקה שהגדרנו בשלב הראשון.

**Private != Secret:**

* כשאנו משתמשים במילה private זה לא כמו להשתמש במילה secret – אנשים חושבים שכאשר אנו משתמשים במשתנה פרטי אז הידע שנשמר בו הוא מוגן, אבל זה לא נכון – לא מומלץ לשמור מידע רגיש במשתנים פרטיים.
* יש מנגנונים בג'אווה שמאפשרים לעקוף את המנגנון של הprivate ומאפשרים לגשת אליו
* השימוש בפרטי הוא כדי לעשות עיצוב יותר טוב.

**2.2 Mini JavaStream: A Little bit More Java:**

**מערכים דו-מימדיים:**

* נערך דו מימדי נגדיר כמערך של מערכים - <type>[][]
* נחשוב כל המערך הדו מימדי כמערך של שורות
* בזיכרון נקבל מערך, שכל ערך בו הוא רפרנס למערך נוסף.
* הגדרת גודל המערך - <type>[][] arr = new <type>[row\_num][col\_num]
* אתחול המערך:
  1. אפשר לאתחל איבר איבר – arr[i][j] = x;
  2. אתחול מהיר - <type>[][] arr = { {x,y,z}, {a,b,c} }
* איך עוברים על מערך דו מימדי – נרוץ בלולאה אחת על השורות (על מערך השורות) ובתוכה בלולאה על העמודות (אורך המערך הנוכחי) –

for (int row = 0 ; row < arr.length ; row++) {

for (int col = 0 ; col < arr[row].length ; col++) {

}

}

* אם בהצהרה על המערך הדו מימדי אנו נותנים את מספר השורות ולא נותנים את מספר העמודות נוכל לבחור שכל איבר במערך השורות יהיה באורך אחר.

נגדיר את המערך ע"י <type>[][] arr = new <type>[row\_num][]

לאחר מכן עבור כל שורה נבחר את האורך – למשל נעבור בלולאה על השורות ונבצע

arr[row] = new <type>[col\_num]

**Enum:**

* תרחיש שחוזר על עצמו הרבה הוא משתנה שמקבל אחד ממספר סופי של ערכים.
* Enum זו מילה שמורה.
* הגדרת enum:

enum X {BLACK, WHITE }

* השמות של הערכים שהוא יכול לקבל יהיו באותיות גדולות (כמו קבועים)
* למעשה הערכים יותאמו למספרים יתחילו מ-0 והלאה, כל פעם עולה באחד.
* ערך ברירת המחדל של enum הוא null
* ניתן ליצור מערך של enumים –

EnumName[][] a = new EnumName[][];

* גישה לערך בenum – EnumName.val

**אופרטורים לוגיים:**

* אם תנאי לא מתקיים – if (!cond) {}
* אם שני תנאים מתקיימים – if (cond1 && cond2)
* אם תנאי 1 או תנאי 2 מתקיים – if (cond1 || cond2)
* אפשר לשמור משתנה בוליאני לפי תנאי – Boolean cond3 = !cond1 && cond2 (קודם ! ואז &&)
* מתי ג'אווה לא מחשבת ביטוי:
  1. אם ג'אווה מקבלת שני תנאים שהיא יודעת שאחד מהם הוא בהכרח שקר אז היא לא מחשבת את הביטוי השני בתנאי כי התוצאה תמיד תהיה שקר
  2. בדומה עבור או – אם אחד הוא אמת
* לאופרטור השלילה יש עדיפות

\*\* אם אנו מקבלים למתודה במחלקה מסוימת משתנה בשם של שדה מחלקה ונרצה לבצע השמה נצטרך להתייחס לשדה כ-this.name