Class Notes

קומפיילר

מנגנון ההופך את הקוד שהמתכנת כתב לתוכנית סופית. יש שפות שיש להן קומפיילר (Java, C#, C++, ועוד) ויש שפות שאין להן קומפיילר (JavaScirpt, PHP...). C# מייצרת לנו קובץ הרצה. סיומת הקובץ: exe. הקומפיילר זה המנגנון שהופך את הקוד של המתכנת לקובץ ההרצה. שפות ללא קומפיילר אין להן קובץ הרצה סופי. לדוגמה JavaScript – הדפדפן מריץ את הקוד שורה שורה.

שלושת הקטגוריות של שגיאות בפיתוח תוכנה:

א. שגיאת קומפילציה – Compilation Error

כל שגיאה שמנגנון הקומפיילר מגלה.

מנגנון הקומפיילר מגלה שגיאות תחביר, שגיאה בשפה עצמה, שימוש במשתנה שלא הוגדר וכדומה.

כל שגיאה כזו הקומפיילר מיד מראה + נותן הסבר.

אם יש שגיאת קומפילציה, מנגנון הקומפיילר לא יבנה את התוכנית הסופית ולכן התוכנית לא יכולה לרוץ.

ב-JavaScript אין מנגנון קומפיילר.

ב. שגיאת ריצה – Runtime Error נקראות גם Exceptions (חריגות)

כל שגיאה שגורמת לקריסה בתוכנית לאחר שהתוכנית רצה.

קריסת תוכנית זה אומר – התוכנית מפסיקה מידית לרוץ ומציגה את השגיאה.

ב-JavaScript השגיאה תוצג ב-Console.

לדוגמה ב-C#, ניסיון לקרוא קובץ שלא קיים.

ג. שגיאה לוגית – Logic Error

שגיאה באלגוריתם שלא גרמה לקריסת התוכנית.

לדוגמה, ריצה על מערך ציונים לחישוב ממוצע הציונים,

ביצוע סכום של כל הציונים אך החלוקה היא בגודל המערך פחות 1.

נניח ריצה על מערך בגודל 10, ביצוע סכום, חלוקת הסכום ב-9 מסיבה כלשהי. זה באג לוגי.

במקום הצגת ממוצע א' נכון, יהיה הצגת ממוצע ב' שאינו נכון.

מנגנון Exception Handling

מנגנון טיפול בחריגות.

מכיל את הפקודות הבאות:

א. try – בלוק המכיל את הקוד שעלול לקרוס. לרוב כל קוד הפונקציה אנו מכניסים לבלוק try אם אנו מבצעים טיפול בחריגות.

ב. catch – בלוק שבא מיידית אחרי try, מכיל קוד לטיפול בבעיה.

ג. finally – בלוק שניתן לכתוב אחרי ה-catch והוא מכיל קוד שתמיד יתבצע – גם אם היתה קריסה, גם אם לא היתה קריסה.

השימוש המרכזי הוא ביצוע קטע קוד שתמיד יתבצע – בין אם היתה חריגה ובין אם לא היתה חריגה. לדוגמה, סגירת משאבים, סגירת טיימרים, סגירת קישורים למסדי נתונים וכדומה.

אם נכתוב קוד אחרי ה-try-catch (ללא finally) זה נראה שהוא תמיד יתבצע, אך זה לא נכון.

יש שני מקרים בהם קוד כזה לא יתבצע וקוד שנמצא ב-finally כן יתבצע:

- אם יש return בבלוק ה-try ובבלוק ה-catch.
- אם קרתה חריגה בבלוק ה-catch.

אם הקוד ב-try לא קורס – ה-try יסתיים בהצלחה ואנו לא ניכנס לבלוק ה-catch

אם הקוד ב-try קרס – אנו ניזרק אוטומטית לבלוק ה-catch ונדלג על כל שאר הקוד ב-try שעדיין לא הגענו אליו.

בבלוק ה-catch אנו תמיד נטפל בבעיה שקרתה – הצגת שגיאה למשתמש, או, שמירת השגיאה בקובץ log, או, ביצוע כל דבר אחר לצורך טיפול בבעיה.

ד. throw – מילה שמורה שזורקת חריגה!

מילה זו גורמת לחריגה – שגיאת ריצה!

eval

פונקציה של JavaScript המקבלת מחרוזת המכילה קטע קוד ומריצה את קטע הקוד הזה.

שתי הסיטואציות בהן בלוק Finally יבצע את הקוד שיש בתוכו כאשר קוד הנמצא מעבר לבלוק ה-catch לא יתבצע:

א. פונקציה המכילה return:

```
function getSomeValue() {
  try {
    //...

    return someFirstValue;
  }
  catch(err) {
    // ...

    return someSecondValue;
  }
  finally {
    // this code always executes...
  }

  // this code will never execute!
}
```

ב. קריסה שהתרחשה בתוך בלוק ה-catch (מקרה חמור בפני עצמו):

```
function someFunction() {
  try {
    //...
  }
  catch(err) {
    // ... some crash here!
  }
  finally {
    // this code always executes...
  }
```

```
  // this code will never execute!

}
```