1. ❓ מתי להשתמש ב־ abstract classומתי ב־interface?

🟢 תשובה מדויקת:

אשתמש ב־ interfaceכשאני רק רוצה להגדיר "חוזה" – אילו פעולות מחלקה חייבת לממש, בלי שום מימוש בפועל.
לעומת זאת abstract class, מאפשרת גם להגדיר **פונקציות עם מימוש חלקי**, **בנאים**, **שדות**, ואפילו **stateפנימי** –כך שהיא מתאימה כשיש **קוד משותף** לכל המחלקות היורשות.

אפשר גם לרשת רק מחלקה אחת (single inheritance) אבל אפשר לממש כמה interfaces בו זמנית.

📌 **שורה תחתונה:**

- אם את צריכה היררכיה עם קוד משותף → abstract class
- אם את רק רוצה להבטיח שמישהו יחשוף פעולות מסוימות → interface

---

2. ❓ מתי להשתמש ב־ Compositionומתי ב־Inheritance?

🟢 תשובה מקצועית:

אשתמש ב־ **Composition**כשאני רוצה שהמחלקה "תכיל" רכיב אחר – לדוגמה Car, מכילה Engine. זה מאפשר לי **לשלב תכונות** בלי להיות תלוי במבנה היררכי קשיח.

אשתמש ב־ **Inheritance**רק כשיש ממש קשר של is-a – לדוגמה Dog, הוא סוג של Animal. ירושה קושרת מחלקות חזק מאוד, ולכן עדיף לשמור אותה למצבים ברורים.

3. ❓ מה היתרון של?Polymorphism

🟢 תשובה:

פולימורפיזם מאפשר לי להשתמש במחלקות שונות עם ממשק משותף דרך אותו ממשק – לדוגמה, רשימת Shapeשמכילה Circle, Rectangle, ו־Triangle ולקרוא על כולם shape.Draw() בלי לדעת את הסוג המדויק.

זה **מפשט את הקוד** ומאפשר **הרחבה עתידית** בלי לגעת בקוד קיים – שזה חלק חשוב מעקרון ה־ OCP (Open/Closed Principle).

4. ❓ מה קורה אם אני לא אשתמש ב־virtual / override?

🔴 שאלה מצוינת שדורשת דיוק:

אם לא תשתמשי ב־ virtualעל המתודה במחלקת האב – לא ניתן יהיה **לדרוס** אותה (override) במחלקת הבת.

ואם תנסי להשתמש ב־ overrideעל מתודה **שלא מוגדרת כ־ virtual**, זה יגרור שגיאה בקומפילציה.

אם תגדירי מתודה במחלקת הבת **באותו שם**, אבל לא תשתמשי ב־override (למשל רק public void Method()), זה ייצר – **hiding** כלומר, קריאה דרך מחלקת האב תקרא למימוש של האב (ולא של הבת). זה מקור ל־bugs.

5. ❓ מה ההבדל בין Overloading ל־Overriding?

🟢 תשובה:

- **Overloading** = מתודות באותו שם אך חתימה שונה (כמות/סוג פרמטרים), באותה מחלקה.
- **Overriding** = מחלקת בת משנה התנהגות של מתודה שהוגדרה במחלקת האב (עם override).

6. ❓ מה זה Sealed class או sealed method?

🟢 תשובה:

- sealed class – מחלקה שלא ניתן לרשת ממנה.
- sealed על מתודה – מונע ממחלקה יורשת לדרוס את המתודה הזו, גם אם היא override.

7. ❓ מה זה Constructor Chaining?

🟢 תשובה: שימוש ב־ this(...): כדי לקרוא לקונסטרקטור אחר באותה מחלקה.

9. ❓ מה זה IS ו־AS ב־C#?

🟢 תשובה:

- is בודק האם מופע הוא מסוג מסוים:

```
if (obj is Dog) { ... }
```

- as מנסה להמיר מופע לטיפוס, אם אפשר:

```
Dog d = obj as Dog;
if (d != null) { ... }
```

10. ❓ מה ההבדל בין abstract class ל־interface ב־C# 8+?

🟢 תשובה: בגרסאות חדשות של C#, גם interface יכול לכלול **default implementation** של מתודות.
אבל:

- עדיין אין בו **state** או **שדות**.
- לא ניתן להוסיף **בנאי**.

ולכן עדיין כשיש קוד משותף רציני – עדיף abstract class.

11. ❓ מה זה new keyword על מתודה במחלקה יורשת?

🟥 שאלה טריקית!

אם תכתבי מתודה באותו שם בלי override, ובלי שהאב סימן אותה כ־ virtual, C# תיתן אזהרה.
אם תוסיפי new, את **מסתירה במכוון** את מתודת האב:

12. ❓ מה זה OCP ו־SRP (SOLID) מתוך?

🟢 תשובה:

- SRP – Single Responsibility Principle – כל מחלקה אחראית רק על דבר אחד.
- OCP – Open/Closed Principle – הקוד פתוח להרחבה, סגור לשינויים.
  למשל: הוספת מימוש חדש של interface מבלי לשנות קוד קיים.

13. ❓ מה זה Dependency Injection ולמה זה חשוב?

🟢 תשובה בסיסית:

זה תהליך שבו אני לא יוצרת אובייקטים בעצמי בתוך מחלקה, אלא מקבלת אותם מבחוץ (דרך קונסטרקטור או פרופרטי).
זה הופך את הקוד ל־**גמיש**, **ניתן לבדיקה ולשינוי**, ומפחית תלות בין מחלקות.

14. ❓ מה זה static class? אפשר לרשת ממנה?

🟢 תשובה:

- static class היא מחלקה שלא ניתן ליצור ממנה מופעים.
- כל המתודות בתוכה **חייבות להיות static**.
- אי אפשר לרשת ממנה או לרשת אותה.

15. ❓ מה ההבדל בין override ל־ new ל־ virtual ל־abstract?

הערות	שימוש	מילת מפתח
חייבת להיות במחלקת האב	מצהיר על מתודה שניתן לדרוס במחלקה יורשת	virtual
חייב להיות התאמה מדויקת לחתימה	דריסה של מתודה virtual מהאב	override
רק במחלקה abstract	מתודה בלי מימוש, חייבת להיות ממומשת במחלקה יורשת	abstract
מבלבל, לא מומלץ	hiding – מסתיר מתודה של האב, לא דריסה אמיתית	new

🔄 **למה צריך Constructor Chaining?**

כשיש כמה בנאים עם לוגיקה שונה, אבל רוצים **למנוע שכפול קוד**.

---

1. ❓ האם מחלקה יכולה לרשת מחלקה אחת וגם לממש interface?

✅ **נכון.**

```
public class MyClass : MyBaseClass, IMyInterface
```

#C תומכת רק **בירושה אחת של מחלקה**, אבל **בכמה interfaces במקביל**.

2. ❓ האם מחלקה יכולה לרשת מ־ abstract וגם לממש interface?

✅ **כן.**
זה מצב קלאסי:

```
public class MyClass : MyAbstractBase, IMyInterface
```

אין שום מניעה בזה. להפך – לפעמים זה מבנה מומלץ.

3. ❓ האם interface יכול להכיל (fields שדות?)

✅ **לא.**
עד → 8 #C רק חתימות של מתודות, בלי שדות בכלל.
ב־ +8 #C יש כבר default implementation, אבל עדיין **בלי שדות או state פנימי**.

4. ❓ האם interface יכול להכיל מימושים?

🔶 **תלוי בגרסה.**

- עד ❌ C# 7 ← לא
- מ־ ✅ C# 8 ← כן, מותר default implementation

5. ❓ מה קורה אם מחלקת האב מכילה רק קונסטרקטור פרטי?

🔴 **אי אפשר לרשת ממנה.**

אם למחלקה יש רק – private constructor היא נועדה **למנוע יצירת מופעים**, ולכן גם **אי אפשר לרשת ממנה**.

שימוש טיפוסי static class: או Singleton.

6. ❓ האם אפשר לקרוא ל־ base() מקונסטרקטור?

✅ כן.

```
public class Child : Parent
{
public Child() : base("some param") { }
}
```

🟢 **מתי?**
כשאת רוצה להעביר ערכים למחלקת האב (ולא לשכפל קוד), או כשיש לוגיקה משותפת במחלקת האב.

7. ❓ יש Animal, Dog, Cat, ועכשיו צריך Fish עם לוגיקה שונה. איך מוסיפים בלי לשנות קוד קיים?

🔵 **פתרון קלאסי לפי: OCP (Open/Closed Principle)**

- לא נוגעים ב־Animal
- יוצרים מחלקה Fish : Animal
- אם ההתנהגות שונה מאוד (למשל שחייה במקום הליכה), נוציא אותה לממשק:

8. ❓ האם אפשר לרשת מ־ abstract class מבלי לממש את המתודות האבסטרקטיות?

🟡 **רק אם את עצמך abstract!**

9. ❓ האם אפשר לשלב abstract עם static?

🔴 **לא.**

- abstract דורשת שנממש את המתודה במחלקת הבת
- static אומר שאין מופע בכלל
  → **סתירה גמורה.**

10. ❓ מה זה Strategy Pattern?

🟢 תבנית עיצוב OOP קלאסית.

מאפשר להחליף **התנהגות בזמן ריצה** בלי לשנות את המחלקה.

12. ❓ האם אפשר ליצור interface שמורכב מממשקים אחרים?

✅ כן! קוראים לזה **interface inheritance**

" ❓ אבל אם אי אפשר ליצור מופע מ־ interface – אז למה בכלל לעבוד עם interface ולא עם מחלקה רגילה"?

🟢 תשובה:

כי זה מאפשר לי **לתכנת מול abstraction**, לא מול מימוש קונקרטי. ואז אני יכולה להחליף את המימוש מתי שאני רוצה – בלי לשנות את הקוד שמשתמש בו.

1. ❓ מה ההבדל בין C# ל־Java?

🟢 תשובה:

- שתיהן שפות מונחות עצמים מאוד דומות תחבירית.
- C#היא שפה של מיקרוסופט, מתממשקת מצוין עם Windows ו־.NET.
- Javaהיא שפה של Oracle, מתממשקת עם JVM ורצה על כל מערכת הפעלה.
- ל־ C#יש תכונות מתקדמות כמו LINQ, Events, async/await מובנה.
- ל־ Javaיתרון בגמישות בין־מערכתית (cross-platform) בזכות JVM.

2. ❓ מה ההבדל בין C# ל־JavaScript?

🟢 תשובה:

- C#היא שפה **strongly typed**, מונחית עצמים ומקומפלת – רצה בשרת או על .NET.
- JavaScriptהיא שפה **weakly typed**, דינאמית, רצה בדפדפן (או Node.js).
- ב־ C#נדרש קומפילציה. ב־ JSהקוד מתפרש בזמן ריצה.
- JavaScriptנפוצה יותר בצד לקוח (Frontend), C# בצד שרת (Backend).

3. ❓ מה ההבדל בין Python ל־C#?

🟢 תשובה:

- Pythonשפה דינאמית, קריאה, עם תחביר פשוט, מעולה לסקריפטים Data Science, ו־ Automation.
- C#היא שפה מקומפלת, strongly typed, מעולה לפרויקטים גדולים, עם ביצועים טובים.
- Pythonנוחה מאוד לפיתוח מהיר, אבל פחות בטוחה לסביבות קריטיות בלי בדיקות קפדניות.
- C#עדיפה לאפליקציות תעשייתיות, משחקים (Unity), מערכות מורכבות ו־ Webעם ASP.NET.

4. ❓ מה ההבדל בין TypeScript ל־JavaScript?

🟢 תשובה:

- TypeScriptהיא JavaScript **עם טיפוסים – (types)** כלומר שפה עם **static typing**.
- היא מאפשרת לזהות שגיאות בזמן קומפילציה במקום רק בריצה.
- הקוד ב־ TypeScriptעובר **transpile** ל־ JavaScript (רגיל) הדפדפן מבין רק JS).

- TypeScriptעוזרת בניהול פרויקטים גדולים עם מחלקות, ממשקים, טיפוסים, וכו.'

5. ❓ האם JavaScript היא שפה מונחית עצמים?

🟡 תשובה מדויקת:

כן, אבל לא כמו C#/Java.
ב־JS יש **object-based programming** עם **prototypes** (במקום מחלקות) לפני ES6.
ב־ES6 הוסיפו class־ו־extends, אבל מתחת זה עדיין Prototype-based.

6. ❓ שאלה פתוחה: באיזו שפה תבחרי לבנות מערכת בק־אנד של מערכת ניהול חשבוניות?

🟢 תשובה חכמה:

תלוי בדרישות.
אם זו מערכת מבוססת Windows/Enterprise → אבחר C# עם ASP.NET.
אם זו מערכת פשוטה עם הרבה לוגיקה טקסטואלית ו־OCR → ייתכן ש־Python תתאים יותר (למשל עם Flask).
אם הצוות כבר משתמש ב־Node.js גם לפרונטאנד → אולי נעדיף JS/TS ל־Backend עם Express.

7. ❓ איזו שפה מהירה יותר Python – או C#?

🟥 תשובה טכנית:

בדרך כלל C# **מהירה משמעותית יותר** מ־Python בגלל שהיא **שפה מקומפלת**, בעוד Python **מתפרשת בזמן ריצה**.
עם זאת, לפעמים מה שחשוב זה זמן פיתוח ולא זמן ביצוע – ושם Python מנצחת.