**הרכבת מחלקות**

**ב-Java, C# אנחנו בדרך כלל נעשה הקצאה דינמית (או שלא) בבנאי של המחלקה המורכבת. בשפות הללו המופע של המחלקה המורכבת מקצה למחלקות המשנה רק מצביעים.**

**ב-C++ לעומת זאת, (אלא אם כן נגדיר במפורש מצביעים) המופעים של משתני המחלקות המרכיבות מוקצים אוטומטית עם המופע של המחלקה המורכבת.**

**בבנאי של מחלקה מורכבת ב-C++ המתכנת חייב לדאוג לקריאה הבנאים הנכונים של מחלקות המשנה. הדבר נעשה ע"י רשימות אתחול לפי שמות המשתנים.**

**חריגות ב-C++**

**מושג יחסית חדש.**

**חריגות ב-C++ הם כמעט תמיד חריגות של המתכנת.**

**חריגות מערכת אינם ממומשים במנגנון הזה אלא במנגנון של שפת C הנקרא סיגנלים signals.**

**יוצא דופן הוא כישלון של new.**

**גם לא קיימת היררכיה מוגדרת מראש של חריגות.**

**האיבר הנזרק יכול להיות כל סוג משתנה שיש.**

**שדות ומתודות סטטיים**

**אפשר לעבוד ב-C++ בשיטות העבודה של Java ו-C# אבל לא חייבים.**

**קריאה למתודה או שדה סטטי**

**שם המחלקה::שם השדה**

**אתחול של שדה מידע סטטי הוא מחוץ למתודות כלשהן (בדרך כלל בקובץ ה-h)**

**התמרות משתמעות implicit promotions או implicit conversions**

הקומפילר של C++, אם הוא נתקל במתודה **M** שמקבלת פרמטר מסוים מסוג **T**, אם הוא **לא** מוצא מתודה או פונקציה בצירוף הזה **M(T)** לפני שיתייאש, הוא יחפש מחלקה, שיש **בנאי** **שיכול** להיקרא עם **פרמטר יחיד מסוג T**, שלן **יש** מתודה או פונקציה העונה **M(T).**

באים בחשבון בנאים עם פרמטר יחיד או יותר פרמטרים שלכולם חוץ אולי הראשון יש ערך ברירת מחדל.

הבעיה העיקרית: יכול להיווצר קוד דו-משמעי.

אפשר לבטל את המנגנון הזה ע"י כך שכל בנאי פוטנציאלי כזה יוגדר explicit.

ממשקים ומחלקות אבסטרקטיות ב-C++

להלכה אין כאלו דברים ב-C++.

מה שיש הוא להגדיר מתודה

**virtual סוג שם(פרמטרים) = 0;**

הכרזה כזו מגדירה מתודה אבסטרקטית שהופכת את המחלקה לאבסטרקטית.

באופן לא פורמלי, ממשק הוא מחלקה שיש לו רק שדות כאלו.

**הורשה מרובה**

**ב-C++ מחלקה יכולה לרשת ממספר מחלקות.**

**הנוהל ב-Java, C# שבו מחלקה יורשת מחלקה אחת וכמה ממשקים מיושם בצורה הזו.**

**הדבר הזה יכול לגרום לבעיות, לחלק יש פתרון (שאולי) מניח את הדעת ולחלק לא.**

**בעיה ראשונה היא בעיה של "ירושת יהלום":**

**מחלקה יכולה בעקיפין לרשת מאותה מחלקה אחרת פעמיים או יותר.**

**פתרון הוא שכל הורשה תוגדר גם virtual.**

**הורשה מרובה כללי**

**מחלקה יכולה לרשת מכמה מחלקות אמתיות.**

**הבעיה העיקרית (מלבד יהלום) הוא התנגשות שמות, שיכולה להיפתר רק ע"י דריסת השמות המתנגשים וציון מפורש של שמות מחלקות האב, במידה וקוראים להם.**

**תכנות ג'נרי או יותר נכון תבניות templetes ב-C++**

**מימוש טמפלטים ב-C++ נעשה על ידי מנגנון מקרו.**

* **מקרו הוא אמצעי תכנות שבו הקומפילר בשלב מקדים יוצר קובץ מקור חדש מקובץ קיים ורק הקובץ החדש עובר קומפילציה.**
* **השלב הזה נקרא pre-processing או pre-compile.**
* **לעובדה הזו יש השלכות:**

1. **לשימוש מרובה בטמפלטים יש נטייה להגדיל את הקובץ הבינארי הביצועי (.exe) שהיא תוצאת הקומפילציה.**
2. **לא ניתן לממש טמפלטים בקובץ cpp נפרד ולשלב אותו בקומפילציה. אם רוצים לשים בקובץ נפרד, חייבים לעשות לו #include לקבצים האחרים. המסורת היא לקרוא לקובץ הזה עם סיומת .hpp.**

**קוד ג'נרי או טמפלטים הם קוד שמשאיר סוגי שדות פתוח.**

**מבחינים בין שני סוגי טמפלטים:**

* **פונקציות טמפלטיות (אבות טיפוס לאלגוריתמים)**
* **טמפלטים של מחלקות.**

**הגדרה של טמפלט נראית ככה:**

**template <class שם סמלי לסוג משתנה>**

**קוד טמפלטי**

**או**

**template <typename שם סמלי לסוג משתנה>**

**קוד טמפלטי**

**"שימוש" בטמפלט של פונקציה משתמע מהתוכנית אבל לא מצוינת במפורש.**