Checklista för konstruktion av klasser

Konstruktion av klasser

- Deklarera datamedlemmar **private**.

  I en basklass kan **protected**-deklarerade funktioner vara ett sätt ge subklassers medlemsfunktioner effektiv åtkomst till privata datamedlemmar i basklassen men föredra normalt **protected** åtkomstfunktioner.

- Använd inte **friend**-deklarationer.

  Motverkar inkapsling, vilket leder till kod som blir svårare att underhålla.

- Medlemsfunktioner som inte ändrar på objekt ska vara **const**.

  const-deklaration av medlemsfunktioner är mycket viktigt; det framgår tydligt av koden att funktionen inte ändrar objekt; kompilatorn kontrollerar att detta; endast **const**-deklarerade funktioner kan användas på konstanta objekt (i konstant kontext). Medlemsfunktioner som inte ändrar externa egenskaper hos ett objekts men det interna tillståndet ska **const**-deklareras och datamedlemmar som ändras ska deklareras **mutable**.

- Medlemsfunktioner som inte kräver **this** ska vara **static**.

  Om inget objekt behövs ska funktionen inte vara beroende av något objekt för att kunna användas.

- En härledd klass ska ha högst en basklass som inte är en gränssnittsklass (interface class).

  Att härleda från två eller flera basklasser som inte är gränssnittsklasser är sällan korrekt, däremot kan det vara meningsfullt att en klass implementerar fler är ett gränssnitt.

- Överskuggande funktioner ska deklareras **override**.

  Deklarera en polymorf funktion **virtual** endast i den första deklarationen. Använd inte **virtual** i överskuggningar, **override** anger att det är överskuggning och leder dessutom till kontroll av att så är fallet.

- Deklarera destruktorn för en typ som används som basklass **virtual**, **privat** eller **protected**.

  Om objekt kan komma att destrueras via basklasspekare ska destruktorn i basklassen vara **virtual** (och **public**), annars kommer inte destruktorerna för subklasserna att anropas. Om objekt inte kommer att destrueras via basklasspekare bör destruktorn deklareras **protected** eller **private**, vilket ger kompileringsfel om objekt destrueras på ett felaktigt sätt.

- En konstruktor ska initiera alla basklasser och icke-statiska datamedlemmar.

  En konstruktor bör initiera objektet fullständigt och explicit initiering gör att risken för felaktigt tillstånd efter att lyckad konstruktion har genomförts reduceras. En kopierings- eller flyttkonstruktor bör initiera varje icke-statisk datamedlem i medlemsinitierarlistan eller, om det inte är möjligt, i konstruktorkroppen. För andra konstruktorer gäller att varje icke-statisk datamedlem bör initieras på följande sätt, i föredragen ordning: 1) icke-statisk datamedlemsinitierare (NSDMI, non-static data member initializer), eller 2) i medlemsinitierarlistan, eller 3) i konstruktorkroppen. Man bör inte ha både en NSDMI och en medlemsinitierare i en konstruktor. Basklasser kommer alltid att initieras av sin defaultkonstruktor om ingen explicit initierare finns i en subklass.

- Initierarna i en medlemsinitierarlista ska anges i samma ordning som datamedlemmar deklareras.

  Initiering sker alltid i följande ordning, oavsett den ordning i vilken medlemsinitierare skrivs:
  1) virtuella basklasser i ordning uppifrån och ner, vänster till höger, enligt deklarationsordningen,
  2) direkta icke-virtuella basklasser i ordning från vänster till höger, enligt deklarationslistan,
  3) icke-statiska datamedlemmar i den ordning de deklareras i klassen.

- Använd delegerande konstruktorer för att minska duplicering av kod.

- Har du definierat *kopieringskonstruktor destruktor* och *kopieringstilldelningsoperator* för en klass om de kompilatorgenererade versionerna inte fungerar?

  Om en klass har en icke-trivial destruktor ska den normalt också ha användardefinierad kopieringskonstruktor och kopieringstilldelningoperator. En icke-trivial destruktor är exempelvis en som återlämnar dynamiskt minne eller frisläpper andra resurser.

För vissa klasser kan inte kopiering tillåtas av någon anledning och då eliminerar man kopieringskonstruktorn och kopieringstilldelningsoperatorn genom att deklarera dem som *deleted*.

- Om du har definierat en egen kopieringskonstruktor och en egen kopieringstilldelningsoperator är det troligt att det även ska finnas en *flyttkonstruktor* (*move*) och en *flytilldelningsoperator.*
- Har du deklarerat medlemsfunktion som ska ha polymorft beteende (kunna överskuggas) **virtual**?
- Har den översta basklassen i en polymorf klasshierarki en *virtuell destruktor*?
  En klass som har en virtuell medlemsfunktion ska ha en publik virtuell destruktor.
  Även om inte basklassen i sig behöver ha en destruktor måste ändå en virtuell destruktor deklareras för att säkerställa att subklassers destruktorer körs i alla sammanhang och speciellt i polymorfa sammanhang.
  Om klasserna i en klasshierarki aldrig kommer att användas polymorft behövs ingen virtuell destruktor och i sådant fall ska det inte heller finnas några virtuella medlemsfunktioner.
- Har du förhindrat automatisk typomvandling med typomvandlande konstruktorer, om det *inte* är önskvärt (**explicit**)?
  En typomvandlande konstruktor kan anropas med *ett* argument av annan typ än klassen ifråga. Typomvandling från en klass till en annan typ med typomvandlande operatorfunktion ska man vara mycket restriktiv med, sådana ställer ofta till med mer problem än de löser, och det är i detta fall normalt **explicit** som gäller.
  Definiera optimerade versioner av exempelvis överlagrade operatorer för att undvika automatisk typomvandling med tillhörande temporära objekt i typblandade uttryck.
- Följer egendefinierade operatorfunktioner den semantik som gäller för motsvarande inbyggda operator?
  Tilldelningsoperatorer, exempelvis, ska returnera en referens till vänsteroperanden.

Tumregel för deklaration av de sex speciella medlemsfunktionerna:

- om en speciell medlemsfunktion ska finnas och den kan genereras korrekt av kompilatorn, *defaulta* den.
- om en speciell medlemsfunktion **inte** ska finnas men den genereras av kompilatorn, *deleta* den.
- om en speciell medlemsfunktion **inte** ska finnas och den **inte** genereras av kompilatorn, deklarera den inte alls.

Vanliga fel i samband med klasser

Olika översättningssystem för C++ kan detektera ett visst fel i olika faser av översättningen, antingen under kompileringen, fel meddelas vanligtvis i termer av ”`error:`…”, eller vid länkningen, fel meddelas i termer av ”`ld:`...” eller ”`ild:`…” (*incremental*).

- *Kompilatorn/länkaren säger att definitionen för en deklarerad medlemsfunktion saknas*.
  Det kan stämma – du har helt enkelt glömt bort den separata definitionen.
  Det finns en definition – varför hittas den inte?
  Om definitionen finns på en separat implementeringsfil kan problemet bero på att du har glömt att ta med implementeringsfilen i kompileringen/länkningen.
  Du kan ha glömt klassprefixet framför namnet i definitionen.
  Om funktionen var **const**-deklarerad i deklaration kan du kan ha glömt **const** i definitionen (**const** är särskiljande vid överlagring och överridning).
  Det kan finnas någon annan avvikelse i funktionshuvudena för deklarationen och definitionen, till exempel i parameterdeklarationerna (antal, typ, ordning).
- *Kompilatorn/länkaren säger något om virtual table,* `__vtable`*, eller liknande*.
  Gäller något fel kopplat till virtuella medlemsfunktioner. Felet kan i många fall vara av samma slag som anges under punkten ovan, dvs av någon anledning kan inte en definition som motsvarar en viss deklaration hittas.

- *Kompilatorn signalerar fel i en deklaration, till exempel i en deklarationen av en parameterlista och du ser absolut inget fel och speciellt inte där kompilatorn anger att det skulle vara fel.*

  Det kan bero på att någon parameter är av standardbibliotekstyp, exempelvis `string`, och du har glömt att inkludera motsvarande ”header” (`<string>`), eller glömt att öppna namnrymden `std`, eller glömt att använda namnrymdsprefixet `std::`, exempelvis `std::string`.

- *Programmet kraschar oväntat med exempelvis ett kryptiskt meddelande som ”bus error” eller liknande. Du har svårt att detektera exakt var felet inträffar men det verkar vara kopplat till användningen av klassobjekt (till exempel vid retur från funktion, tilldelning).*

  Det kan hänga ihop med att du har en icke-trivial klass och du har glömt att definiera egna versioner av speciella medlemsfunktioner, till exempel kopieringskonstruktor och kopieringstilldelning, vilket kan leda till minneshanteringsfel, speciellt om en destruktor har definierats.

  Programmet kan ha ”skrivit sönder minnet”, till exempel genom att ha stegat för lång i fält (inte sällan teckenfält) och därigenom ändrat i minne som tillhör en annat objekt.

- *Programmet kraschar med meddelande ”Segmentation fault”.*

  Typiskt relaterat till pekarfel: en tompekare avrefereras, en pekare som används är oinitierad, …

  Programmet kan även ha hamnat i en oändlig rekursion och vilket medfört att exekveringsstacken har förbrukats.

  Operationer på containeriteratorer som pekare ”förbi-slutet” kan leda till ”segmentation fault”.

Det kan skilja en hel del i felmeddelanden mellan olika system och ibland kan kompilering med olika kompilatorer vara till hjälp för att försöka förstå annars svårbegripliga felmeddelanden.

Kolla alltid kompileringsfelen som kommer först i listan – ibland kan det vara enkla fel som genererar mängder av felmeddelanden. Leta efter nyckelord som ”fel”/”error”, ”varning”/”warning”.

Var observant på varningar ("warning:…"), sådana kan i vissa fall peka på saker som utgöra allvarliga fel! Alla varningar bör elimineras.