1. tétel
Betokozás, kivételek kezelése

A, Betokozás: az osztályok bármennyi adatot és tagfüggvényt tartalmazhatnak. Az adatokat el kell rejtenünk, be kell tokoznunk az azokat használó osztály belsejébe. A betokozás nagy előnye, hogy az osztály létrehozója bármikor módosíthatja az osztály belső ábrázolását.
A betokozás lényege egyszerű: úgy kell elképzelni egy osztályt, mint egy 'fekete dobozt', amelynek csak kis része látható. A doboz belsejét nem látjuk, kizárólag a rendelkezésünkre álló felületen, az interfészen, keresztül férhetünk hozzá. Az objektumok használata közben a kód nagyobb része rejtve marad. Általában nem tudjuk, milyen belső adatokat tárol az objektum, és nem is férhetünk hozzá közvetlenül. Az adatokat csak tagfüggvényeken keresztül érhetjük el, amelyek megvédik az osztályt az illetéktelen hozzáféréstől. Ezt az adatok elrejtésének (data hidding, information hidding) nevezzük. A Delphi osztály alapú betokozást valósít meg. Az egységek felületíró (interface) részében meghatározott azonosítók a program többi egységéből is elérhetők, feltéve, hogy a 'uses' utasításban flsoroljuk az azonosítót tartalmazó egység nevét.
Hozzáférési szintek:

1. *private:* utasítással megjelölt tagfüggvények vagy mezők az osztályt bevezető egységen kívül nem érhetők el.
2. *protected:* az adott mező vagy tagfüggvény korlátozott láthatósággal rendelkezik. Kizárólag az adott osztályban, illetve annak leszármazottaiban férhetünk hozzá. Tehát a mező tartalmához csak az osztály, az alosztályok és az egységen belüli kódrészletek férhetnek hozzá.
3. *public:* a program tetszőleges pontján, szabadon hozzáférhetők.

Az osztályok mezőinek általában privátnak kell lenniük, a tagfüggvények többnyire nyilvánosak.
A hozzáférést szabályozó kulcsszavak kizárólag az adott egységen kívüli kóddal szemben védik az osztály felületleíró részben bevezetett tagjait. Ez azt jelenti, hogy ha ugyanabban az egységben határozunk meg két osztályt, nem tudjuk megvédeni azok privát mezőit.

Betokozás tulajdonságokkal: a tulajdonságok használata az OOP lényeges része. A tulajdonságok tulajdonképpen virtuális mezők. Például az alábbi kóddal kiolvashatjuk egy gomb Caption tulajdonságának értékét és felhasználjhatjuk azt egy szövegmező Text tulajdonságának beállításához:

```
Edit1.Text:=Button1.Caption;
```

Úgy néz ki, mintha mezőket írnánk és olvasnánk, a tulajdonságokat azonban közvetlenül összekapcsolhatjuk az adatokkal, illetve az adatok elérését szolgáló tagfüggvényekkel.
A tulajdonság olyan azonosító, amelyet a 'read', illetve a 'write' záradékokon keresztül adatokhoz vagy tagfüggvényekkhez rendelünk.
Pld.: a dátum osztály 'Month' mezőjének meghatározása:

```
Property Month: integer read Fmonth write SetMonth;
```

A 'Month' tulajdonság értékét a program az Fmonth nevű privát mezőből veszi, ha módosítani szerenénk ezt az értéket, a SetMonth tagfüggvény kell meghívnunk. Persze, először meg kell határozni a tagfüggvényeket az osztályon belül.)
Például a Tdate osztály év, hónap és nap elérését szolgáló mezők:

```
Type
    Tdate = class;
  Public
    Property Year: integer read GetYear write SetYear;
    Property Month: integer read GetMonth write SetMonth;
```

Property Day: integer **read** GetDay **write** SetDay;

A hozzá tartozó tagfüggvények:

A, **Function** Tdate.GetYear: integer;
Begin
Result := YearOf (fDate);
End;

B, **Function** Tdate.SetYear (**const** value: integer);
Begin
FDate := RecodeYear (fDate, Value)
End;

(A többi tagfüggvény hasonló ezekhez!!!)

B, Kivételek kezelése: a kivételekkel, melyek módot biztosítanak a hibák és a váratlan helyzetek kezelésére, stabilabbá tehetjük programunkat. A kivételekkel a kód magját teljesen elválaszthatjuk a hibakezelő kódtól, így a program sokkal áttekinthetőbbé válik.
Ha a kódot 'jól írtuk meg', akkor az felismeri a problémát és megpróbálja megoldani a feladatot, ellenkező esetben a kivétel átadódik a hívó kódnak.
Végül, amennyiben kódunk egyetlen része sem tudja kezelni az adott hibát, akkor azt a Delphi fogja kezelni úgy, hogy egy szabványos hibaüzenetet küld, majd megpróbálja folytatni a program futását.
A kivételkezelés kulcsszavai:

1. *try:* ez jelöli ki a védett kód kezdetét,
2. *except:* ez jelöli a védett kódblokk végét. Ezt követik a kivételkezelő utasítások.
3. *finally:* olyan kódblokk, ami akkor is végrehajtódik, ha hiba történt, és akkor is, ha nem történt hiba. Általában olyan műveletek, amit mindenképpen végre kell hajtanunk. Pld.: fájlok vagy adatbázistáblák bezárása.
4. *raise:* mi magunk válthatunk ki kivételeket. A legtöbb kivételt, amelyekkel a Delphi programozása során találkozunk, a rendszer váltja ki, de mi is készíthetünk saját kivételeket a kódban, például, ha érvénytelen vagy egymással összhangban nem álló adatokkal találkozunk futásidőben.

A kivételkezelés nem helyettesítheti a megfelelően megírt vezérlési folyamatot. A felhasználótól érkező adatok ellenőrzéséhez és az egyéb előre látható hibák kiszűréséhez használjuk az 'if' utasítást. Csak a normálistól eltérő vagy előre nem látható helyzetekben használjuk a kivételt.