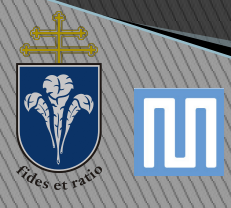
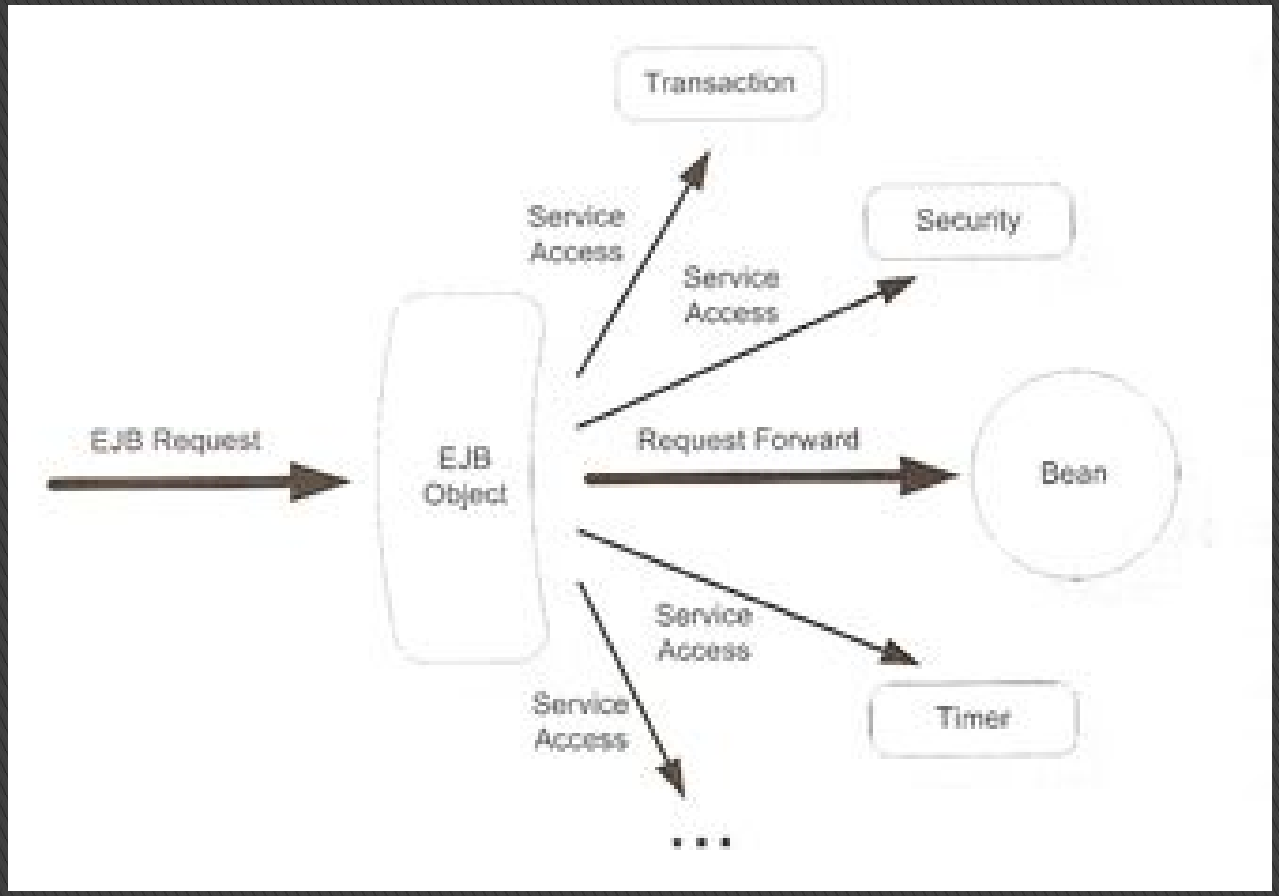
EJB szolgáltatások

EJB Security

- Biztonság
- Tranzakciókezelés
- Timerek
- Interceptorok

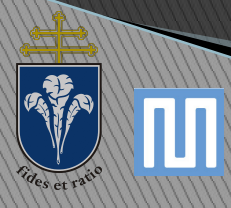


EJB konténer által nyújtott szolgáltatások



EJB Security

- A konténer nyilvántartja a bejelentkezett felhasználókat, azoknak szerepköreit
- Szabályozható, melyik szerepkör milyen oldalakat láthat, milyen függvényeket hívhat meg (későbbiekben részletesebben)
- Hogy melyik EJB függvényt ki hívhatja meg, annotációkkal tudjuk szabályozni



EJB Security példa

```
@Stateless
@RolesAllowed({ADMIN, SUPERUSER})
public class SampleEJB{

    @PermitAll
    public void sampleFunction(){
    }

    @DenyAll
    public void sampleFunction2(){
    }

    @RolesAllowed({ADMIN})
    public void sampleFunction3(){
    }


}
```

Timer Service

- Az üzleti logika megkívánhat időzítést
  - Standard Java-ban jellemző megoldás:

új szál indítása, run() metódus megfelelő felüldefiniálása

  - Java 1.3 óta: java.util.Timer létrehozása és felhúzása, a háttérben ez is egy új szálat indít

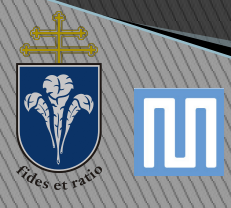
–> EJB-ben tilos új szál létrehozása

- Új megoldás kell, ez a Timer Service

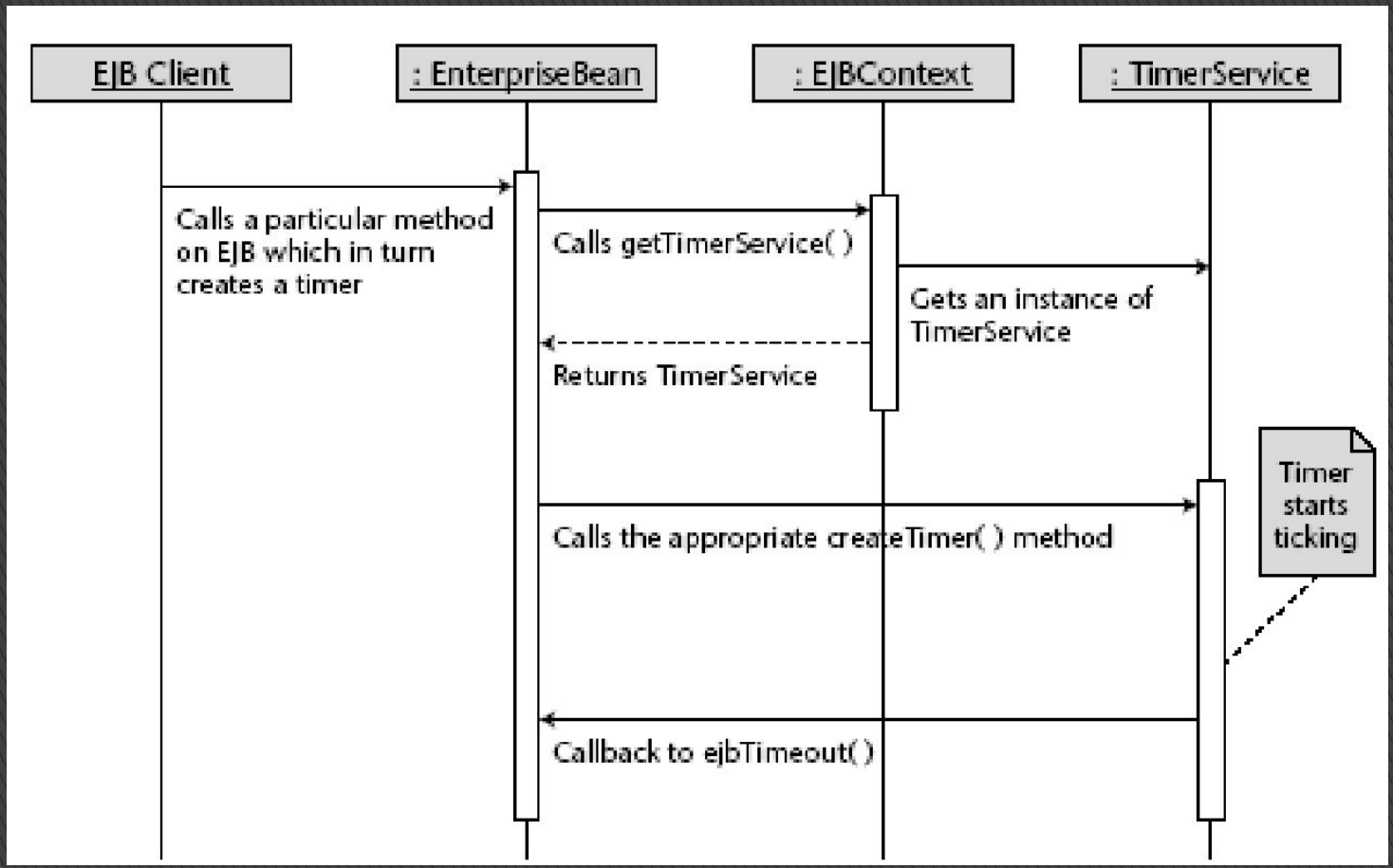
Timer Service

- Saját szál azért nem indítható, mert pl. az EJB konténer nem tudná követni a tranzakciókat

- A konténert kell megkérni, hogy indítson egy timer szálat, így a konténer tudni fog róla
- Az implementációs osztályban pedig az EJBContext a gateway a konténer felé –>

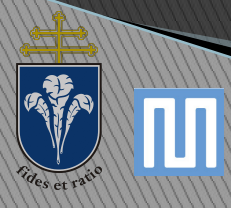


Timer Service



Timer Service

- A Timer indulhat a létrehozás pillanatában, vagy adott idő múlva, vagy adott időpillanatban
- Lejárhat egyszer, vagy bizonyos intervallumonként,
- Állapottal rendelkező session bean nem használhatja
- Lehet perzisztens(szerver leállás/újraindulás után is megmarad – egy példány egy klaszterben)
- A timer lejártakor meghívott callback metódus:
  - TimedObject interfész és ejbTimeout() metódus, vagy
  - @Timeout annotáció (EJB 3 óta)



Timer Service típusai

1. Programozott időzítők
2. Automatikus időzítők

Timer Service (időzítők)

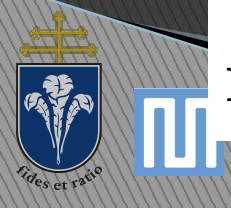
1. **Programozott timerek**: programon keresztül létrehozunk egy időzítőt, amely lejártakor meghívódik a @Timeout–al annotált függvény

```
@Singleton
@Startup
public class TimerServiceImpl {

    @Resource
    private TimerService timerService;

      @PostConstruct
      public void initializeTimer(){
            timerService.createTimer(1000, „timer”);
      }

      @Timeout
      public void timerExpired(Timer timer){
            System.out.println(timer.getInfo());
            System.out.println(„Timer expired”);
      }

}
```

Timer Service (időzítők)

2. **Deklaratív timerek**: @Schedule vagy @Schedules annotációval ellátott függvények előre konfigurált időközönként lefutnak

```
@Stateless
public class AutomaticTimerImpl {


        @Schedule(minute="*/1", hour="*")
        public void runEveryMinute(){
                System.out.println(„this timer runs every minute");
        }

        @Schedule(minute="15", hour="10", dayOfWeek="Mon-Fri  )
        public void runEveryWeekDayAt1015(){
         System.out.println(„this timer runs every work day at 10:15");
        }

        @Schedule(second="*/20", minute = "*", hour = "*"")
        public void runEvery20Seconds (){
                System.out.println(„this timer runs every 20 seconds");
        }


}
```



EJB 3.1 – A Timer Service bővítése

- Többféle időzítési lehetség
- a UNIX cron mintájára, pl. Minden hétköznap, 7, 15 és 20 órakor:

```
@Schedule(hour = "7, 15, 20", dayOfWeek = "Mon–Fri")
```

- Minden 5. percben:

```
@Schedule(minute = "*/5")
```

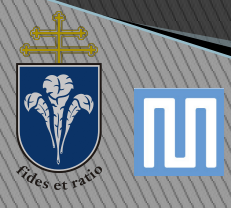
- December utolsó péntekjén, 12 órakor

```
@Schedule(hour = "12", dayOfMonth = "Last Fri",
month="Dec")
```

- ilyen kifejezések programozottan is használhatók, a Timer létrehozásakor adható át egy ScheduleExpression
- @AroundTimeout interceptor

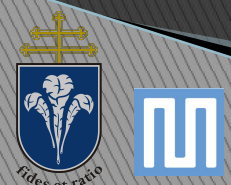
Tranzakciókezelés

- JPA-s témakörnél



Interceptorok

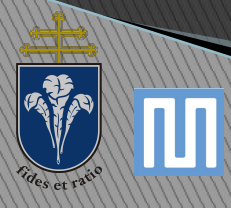
- AOP: Aspect Oriented Programming – bizonyos feladatok azok egész kódot átszövik, érdemes egységesen kezelni, egy helyen definiálni őket
- Metódusok működését tudjuk módosítani, becsomagolni anélkül, hogy a mindenütt módosítanánk a tényleges kódot
- Pl. loggolás – szeretnénk minden függvény futása közben egy bejegyzést írni a log file-ba, hogy a függvény lefutott



Interceptorok

- Az EJB 3 által definiált interceptorok az AOP kezdetleges támogatásának tekinthetők
- Session és message-driven beanekre definiálhatók, vagy egyes metódusokra
- Megszakítják a bean metódusát,

módosíthatják a bemenő paramétereket, akár meg is akadályozhatják a tényleges meghívást

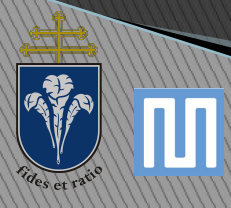


Interceptor hozzáadása EJB-hez

- @Interceptors(MyInterceptor.class) az osztályon vagy a metóduson (class level / method level)
- vagy akár több is hozzárendelhető:

@Interceptors({Interceptor1.class,Interceptor2.class, ...})

- a lista sorrendjében hívódnak meg az interceptorok
- lehet a telepítésleíró xml-ben is konfigurálni (ejb-jar.xml)

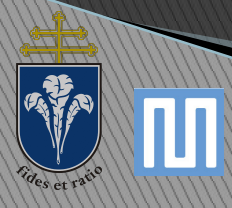


Interceptor írása

- Tetszőleges Java osztály, paraméter nélküli konstruktorral
- Az interceptor metódus:
  - @AroundInvoke-kal van annotálva,
  - dobhat Exception-t
  - használhat függőség injektálást
  - a hívott EJB biztonsági és tranzakciós kontextusában fut
  - bemenő paramétere egy InvocationContext, ezen keresztül éri el a megszakított metódust, paramétereket, a bean példányt, és ebbe rakhat névvel azonosított adatokat más interceptorok számára
- Ha egy EJB callback metódusát külön osztályba rakjuk, akkor azt nem @AroundInvoke-kal, hanem a @PostConstruct, @PrePassivate, stb. annotációkkal kell megjelölni

Interceptor annotációk

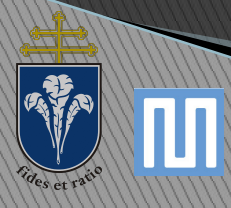
Annotáció	Leírás
javax.interceptor.AroundInvoke	A metódus egy interceptor metódus
javax.interceptor.AroundTimeout	Timer timeout-ja idején hívódik meg
javax.annotation.PostConstruct	Életciklus fázishoz köthető interceptor
javax.annotation.PreDestroy	Életciklus fázishoz köthető interceptor



Interceptorok alkalmazása

Interceptor osztály

```
public class MethodLogger {

    public MethodLogger(){
    };

    @AroundInvoke
      public Object logMethodInvocation(InvocationContext ic) throws Exception {

            System.out.println("Invoking method: "+ic.getMethod());

            System.out.println("Parameters:  "+Arrays.toString(ic.getParameters()));

            return ic.proceed();
      }
}
```



Interceptorok alkalmazása

- Class level interceptor

```
@Stateless
@Interceptors({MethodLogger.class})
public class LoggedEJB {

}
```

- Method level interceptor

```
@Stateless
public class LoggedEJB {

    @Interceptors({MethodLogger.class})
    public void interceptedMethod(){

    }
}
```