Session: 14

EJB Timer Service

Objectives

- Describe how to use a timer in an enterprise application
- Explain the different types of timers in EJB
- Describe how to handle timers in an application
- Describe annotations associated with timers
- Describe operations performed on Timer objects

Introduction

- Enterprise applications require to schedule various tasks such as auditing, generating reports, and so on.
- These tasks are scheduled to be executed at regular intervals or at a prescribed time.
- Java EE provides EJB timers to schedule tasks in the application code.
- EJB TimerService was first introduced in EJB 2.1.
- The TimerService allows the EJB container to register certain EJB methods to be invoked at a scheduled time.

Timer Service 1-2

- The EJB Timer service is a container-managed service.
- `Timer` objects are created by interacting with the `TimerService` of the container.
- Following are the functions which can be implemented by the developer:

Create a timer callback function

- `Timer` object is created with either a calendar time or number of seconds. When the calendar time is reached or when the timer expires, the operation to be performed is defined in the timer callback function.

Process a timer callback function

- Callback method has to be invoked and executed once the timer expires or the scheduled time is reached.

Timer Service 2-2

Check and interact with the timer callback notification object

- When an object generates a timer callback notification, the `TimerService` enables interacting with the notification generating object.

Obtain a list of outstanding timer notifications

- When there are multiple timer callback notifications, the `TimerService` enables listing outstanding timer notifications.

Cancelling a timer notification

- The `TimerService` enables the application to cancel the timer notification according to the application requirement.

Timer Notification

- Timer notification can be defined at the timer expiry event.
- The notification is handled through timer callback methods.
- Developer can use any of the following methods to designate a method as timer notification callback method:
  - Annotate the method with `@Timeout` annotation
  - Implement the `javax.ejb.TimedObject` interface and define `ejbTimeOut()` method
  - Annotate with `@Schedule` annotation

Creating a Timer Callback Notification 1-4

- Following code snippet demonstrates creating and using a `Timer` callback notification:

```
import java.util.Timer;
import java.util.TimerTask;

public class TimerDemo {
  Timer timer;

  public TimerDemo(int seconds) {
    timer = new Timer();
    timer.schedule(new Reminder(), seconds * 1000);
  }
  }
```

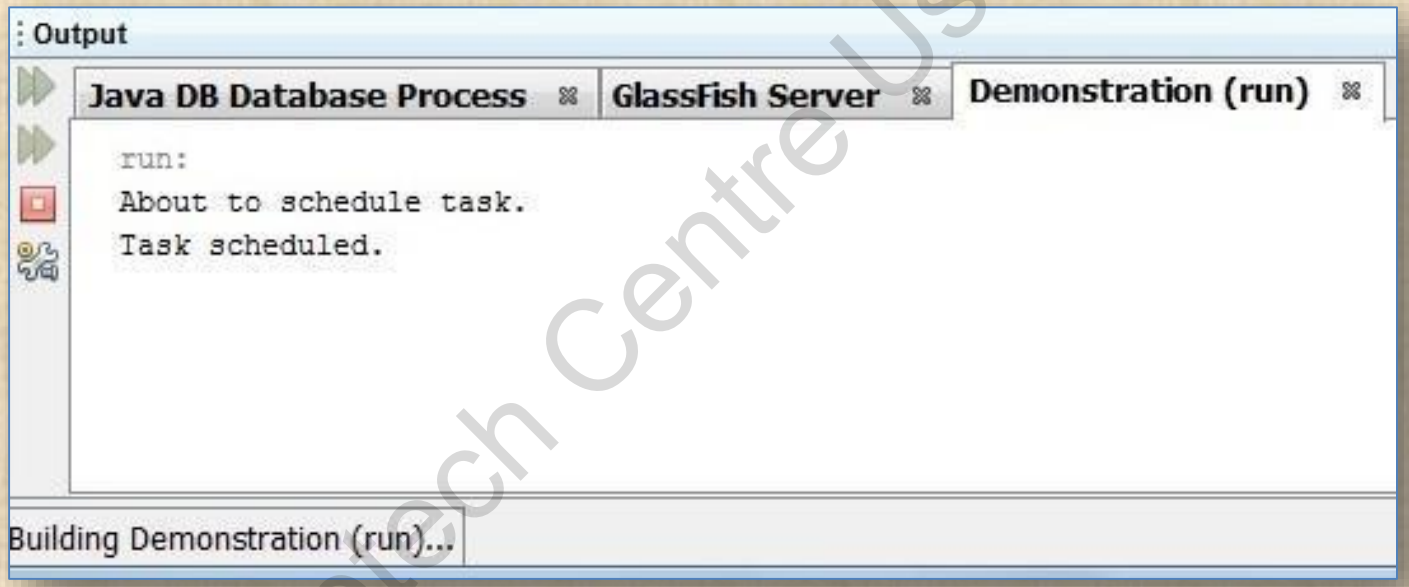

Creating a Timer Callback Notification 2-4

```
 class Reminder extends TimerTask {
    public void run() {
      System.out.println("You have a meeting!");
      System.exit(0);
    }
    public static void main(String args[]) {
     System.out.println("About to schedule a reminder.");
     TimerDemo T = new TimerDemo(10);
     System.out.println("Reminder scheduled after 10
seconds");
   }
}
```

- A `Timer` object is created which expires in 10 seconds.
- The program sends a reminder after the timer expires.
- The `Timer` object is instantiated in the parameterized constructor.
- The timer notification callback method that is an internal class `Reminder` is invoked when the timer expires.

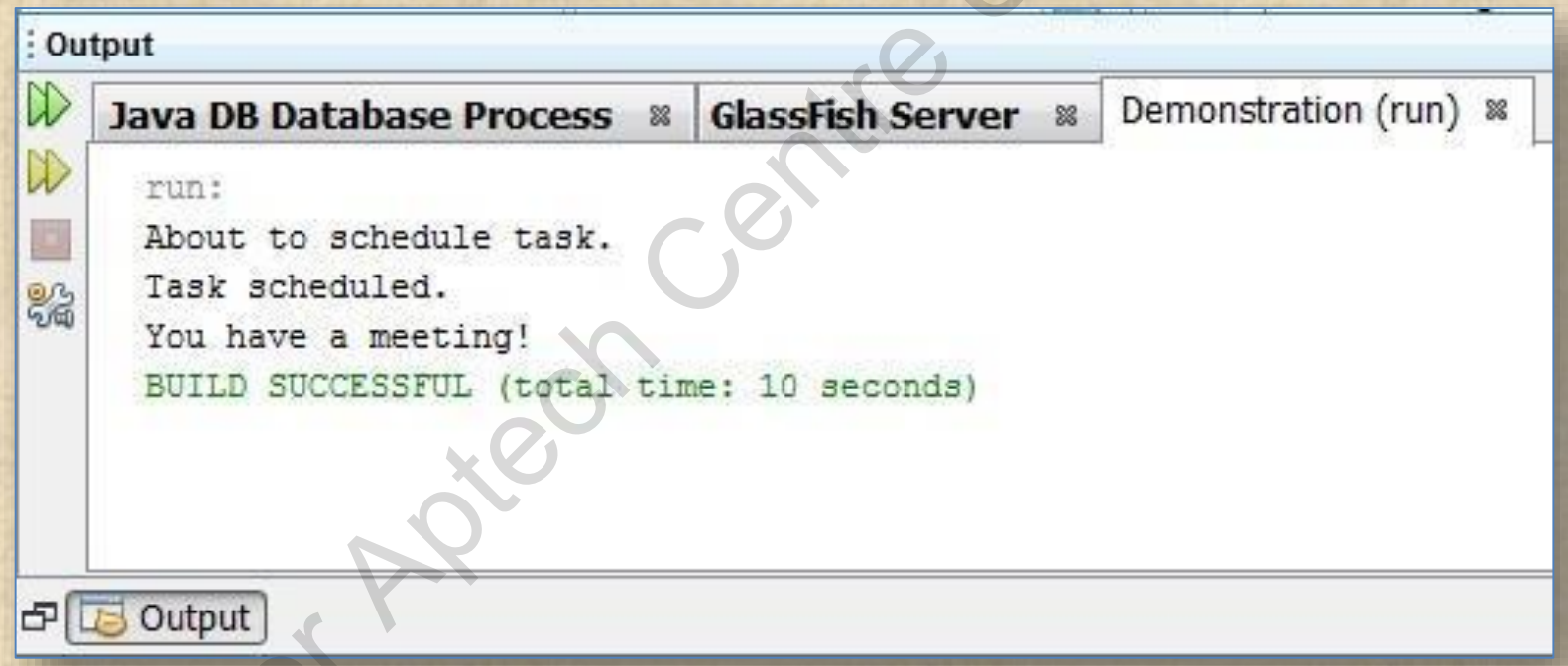
Creating a Timer Callback Notification 3-4

- Following figure shows the output before the timer callback notification:

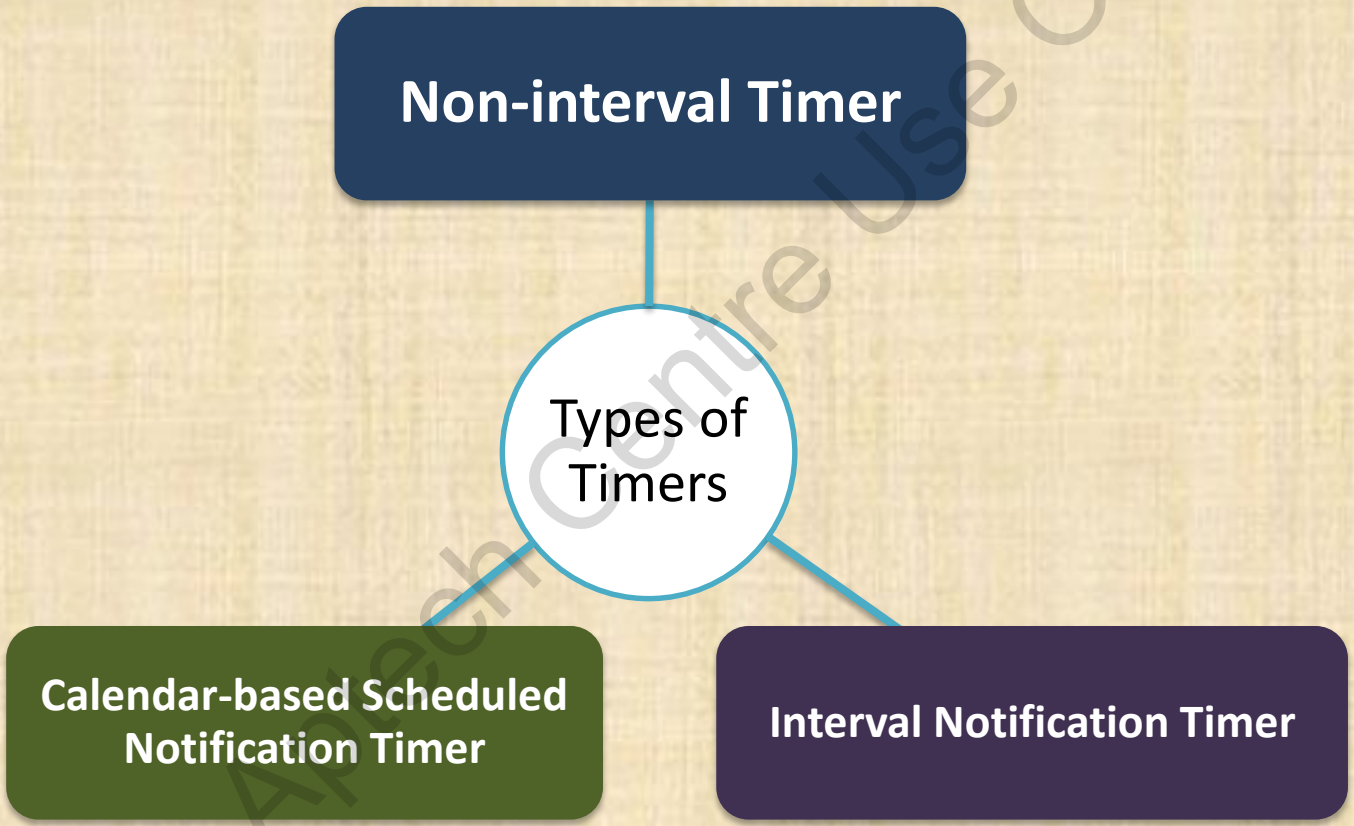


Creating a Timer Callback Notification 4-4

- Following figure shows the execution after the timer of 10 seconds elapses and the timer notification callback method is invoked:



Types of Timers



Non-Interval Notification Timers

- Non-interval notification timers create a notification based on the absolute time or delay interval.
- Notifications are generated based on a `Date` object or delay interval specified in terms of seconds.
- They are non repetitive.
- Two variants of non-interval notification timers:
  - Absolute Time Single Event Timer
  - Relative Time Single Event Timer

Absolute Time Single Event Timer

- Invokes a timer callback method only once at a specific time in the future.
- The timer is created using `createTimer()` method.
- Following is the syntax for `createTimer()` method:

Syntax:

```
Timer createTimer(Date expiration, Serializable
info)
```

- The method returns a `Timer` object, which expires at the date specified as the `Date` parameter.
- The `info` parameter represents the application information to be delivered along with the notification.

Relative Time Single Event Timer

- Defines a timer callback which is invoked after a specific time in the future.
- The timer is created using `createTimer()` method.
- Following is the syntax for `createTimer()` method used to create a relative timer:

Syntax:

```
Timer createTimer(long duration, Serializable info)
```

- The duration parameter is a milliseconds unit.
- Therefore, an appropriate long value has to be specified as the parameter.

Interval Notification Timers 1-2

- The timer notifications are generated at regular intervals.
- Used when certain task has to be executed at regular intervals of time.
- There are two variants of interval notification timers:
  - Interval timer with initial absolute timer
  - Interval timer with initial relative timer

Interval Notification Timers 2-2

Interval timer with initial absolute timer

- First timer notification specified by an absolute value
- Duration of consecutive intervals are specified
- **Syntax:**

```
createTimer (Date initialExpiration, long
interval, Serializable info)
```

Interval timer with relative initial timer

- First timer notification by a relative value, which is relative to the time when the timer is scheduled.
- Consecutive timer notifications are specified through the long value.
- **Syntax**:

```
createTimer(long initialDuration, long interval,
Serializable info)
```


Calendar-Based Notification Timers 1-3

- Calendar-based notification timers generate notifications based on a schedule created on the basis of absolute calendar dates.
- Calendar-based expressions can be used along with `@Schedule` annotation.
- `Schedule` class is used to define the schedule of timer notifications in the application.
- Following is a sample calendar expression:
  - `year = ”2008,2012,2016” dayofWeek = “Sat,Sun” hour = “10” minute = “0-10,30,40”`

Calendar-Based Notification Timers 2-3

- Following table shows the attributes that can be used in calendar expressions:

Attribute	Default Value	Possible Values
`second`	0	[0,59]
`minute`	0	[0,59]
`hour`	0	[0,23]
`dayofWeek`	*	[0-7] or { “Sun”,”Mon”,“Tue”,”Wed”,”Thu”,”Fri”,”Sat”}. Both 0 and 7 refer to Sunday
`month`		[1-12],{”Jan”,”Feb”,”Mar”,”Apr”,”May”,”Jun”,”Jul”,”Aug”,”Sep”,”Oct”, “Nov”,”Dec”}
`dayofMonth`	*	[1-31] or {“1st”,”2nd”,”3rd”…and so on} or “Last” which implies the last day of the month.
`year`	*	Four digit value
`timezone`		Timezone according to tz database

Calendar-Based Notification Timers 3-3

- Following are the different forms in which a Calendar expression can accept data:

As a single value

- `year = "2015" month = "May"`

As a wild card expression

- `month ="*"`

As a list

- `month = "May,Apr,Dec,Nov"`

As a range expression

- `dayofMonth = [10-20]`

As increment expression

- `second = "*/15"`

Creating a Timer Object 1-2

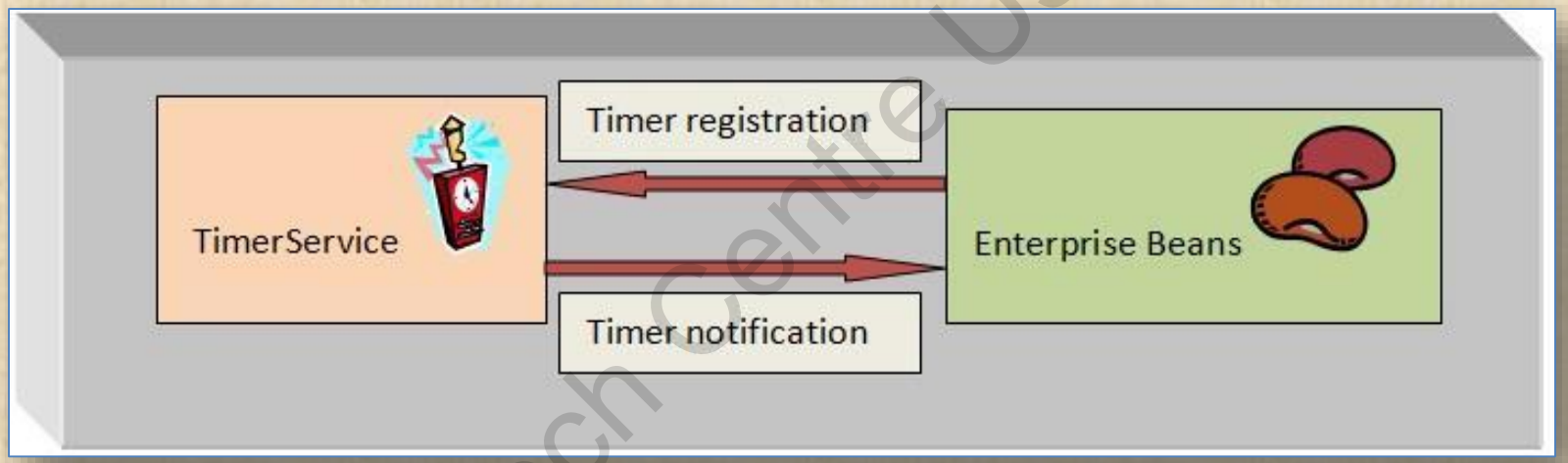
- To create a `Timer` object, the enterprise bean has to create an object reference of `TimerService`.
- Following code snippet demonstrates creating the reference using `getTimerService()` method:

```
private SessionContext SC ;
……..
TimerService TS = SC.getTimerService();
timerService.createTimer(1000*60,text);
. . .
```

- `Timer` objects are also created declaratively with `@Schedule` annotation.

Creating a Timer Object 2-2

- Following figure illustrates how a `Timer` object is created using `TimerService`:



Processing a Timer Callback Notification 1-2

- A timer callback notification can be processed in one of the following ways:
  - Through an enterprise bean which implements `TimedObject` interface
  - Through a method prefixed `@Timeout` annotation
- Through a method annotated with `@Schedule` used to process the timer callback notification.

Processing a Timer Callback Notification 2-2

Using the `TimedObject` interface

- The `TimedObject` interface has only one method `ejbTimeOut()` which accepts the `Timer` object which it has to handle as a parameter.

Using `@TimeOut` annotation

- A timeout callback method is defined in the bean class and annotated with `@Timeout` annotation.

Using `@Schedule` annotation

- The method annotated with `@Schedule` annotation receives notifications from the `TimerService`.

Guidelines for Coding Timer Handler Method

- The developer should consider the following factors while creating timer handler methods:
  - Identifying the timer which has sent a notification
  - Handling application shutdowns
  - Transaction handling for timer notification callback methods
  - Propagating the security context to the timer callback notification
  - Handling non-persistent timers

Managing Timer Objects

- The `Timer` interface provides various methods to manage timer objects in the applications.
- Developers may have to perform the following tasks while managing timers:
  - Interrogate a timer callback notification
  - Obtain a list of outstanding timers
  - Cancel timer notification

Interrogating a Timer Callback Notification

- Following are the methods provided by the `Timer` class for interrogating the `Timer` object:
  - `getHandle()`
  - `getInfo()`
  - `getNextTimeOut()`
  - `getSchedule()`
  - `getTimeRemaining()`

Obtaining a List of Outstanding Timers

- `getTimers()` method of the `TimerService` class is used to retrieve the list of outstanding timers.
- The prototype of `getTimers()` method is:

```
Collection<Timer> getTimers();
```

- `cancel()` method of `Timer` class is used to cancel the timers. If there are no timers existing then the bean would encounter a `NoSuchObjectException`.

Summary

- Enterprise applications use `Timer` objects to schedule tasks in the application.
- Java EE provides `TimerService` class to handle, create, and manage timers in the application.
- Timers can be created declaratively through `@Timeout` and `@Schedule` annotations.
- Timers are created and managed programmatically using `TimedService`, `Timer`, and other classes available in Java EE.
- Timers can be defined as absolute timers, recurrent timers, and calendar based timers.