Distributed Objects and Components

Chapter 8 from the TextBook

Case Study: Enterprise Java Beans

- The advantage of application servers is that they provide comprehensive support for one style of distributed programming – the three tier approach.

- Disadvantages:
  - Prescriptive
  - Heavyweight

-

Case Study: Enterprise Java Beans

- **Enterprise JavaBeans (EJB)** is a specification of a **server-side**, managed component architecture and a major element of the Java Platform

- **Enterprise Edition (Java EE),** is a set of specifications (including RMI and JMS) for **client-server** programming

- EJB is defined as a *server-side component model* where potentially large numbers of clients interact with a number of services realized through components or configuration of components.

Enterprise Java Beans Architecture

- EJB provides direct support for the three-tier architecture
- Container pattern is used to provide support for key distributed system services (like what? How?)
  - Container-managed
  - Bean-managed
- Candidate applications?
  - Tiered applications?
  - Peer-to-peer?
  - Embedded systems?

Content of section "Describing an application server" from Chapter 2: RH student guide removed for Copyright preservation.

Content of section "Describing an application server" from Chapter 2: RH student guide removed for Copyright preservation.
Figure 2.1

Content of section "Describing an application server" from Chapter 2: RH student guide removed for Copyright preservation.

JBoss Enterprise Application Platform (EAP)

Content of section "Describing an application server" from Chapter 2: RH student guide removed for Copyright preservation.

JBoss Enterprise Application Platform (EAP)

Content of section "Describing an application server" from Chapter 2: RH student guide removed for Copyright preservation.

JBoss Enterprise Application Platform (EAP)

Content of section "Describing an application server" from Chapter 2: RH student guide removed for Copyright preservation.
Figure 2.2

Containers

- A container is a logical component within an application server that provides a runtime context for applications deployed on the application server.

- A **container acts as an interface** between the application components and the low-level infrastructure services provided by the application server

- Containers are responsible for security, transactions, JNDI lookups, and remote connectivity and more.

EJB Component Model

- A *bean* in EJB is a component offering one or more **business interfaces** to potential clients of that component

- Interfaces can be either *remote*, requiring the use of appropriate communication middleware (such as RMI or JMS), or *local*, in which case more direct, and hence efficient, bindings are possible

- EJB supports required interfaces (how?)

EJB Component Model

- Two main styles of beans are supported in the EJB specification

- Session beans: A session bean is a component implementing a particular task within the application logic of a service
  - Stateful
  - Stateless
  - Singleton

- Message-driven beans support indirect communication.

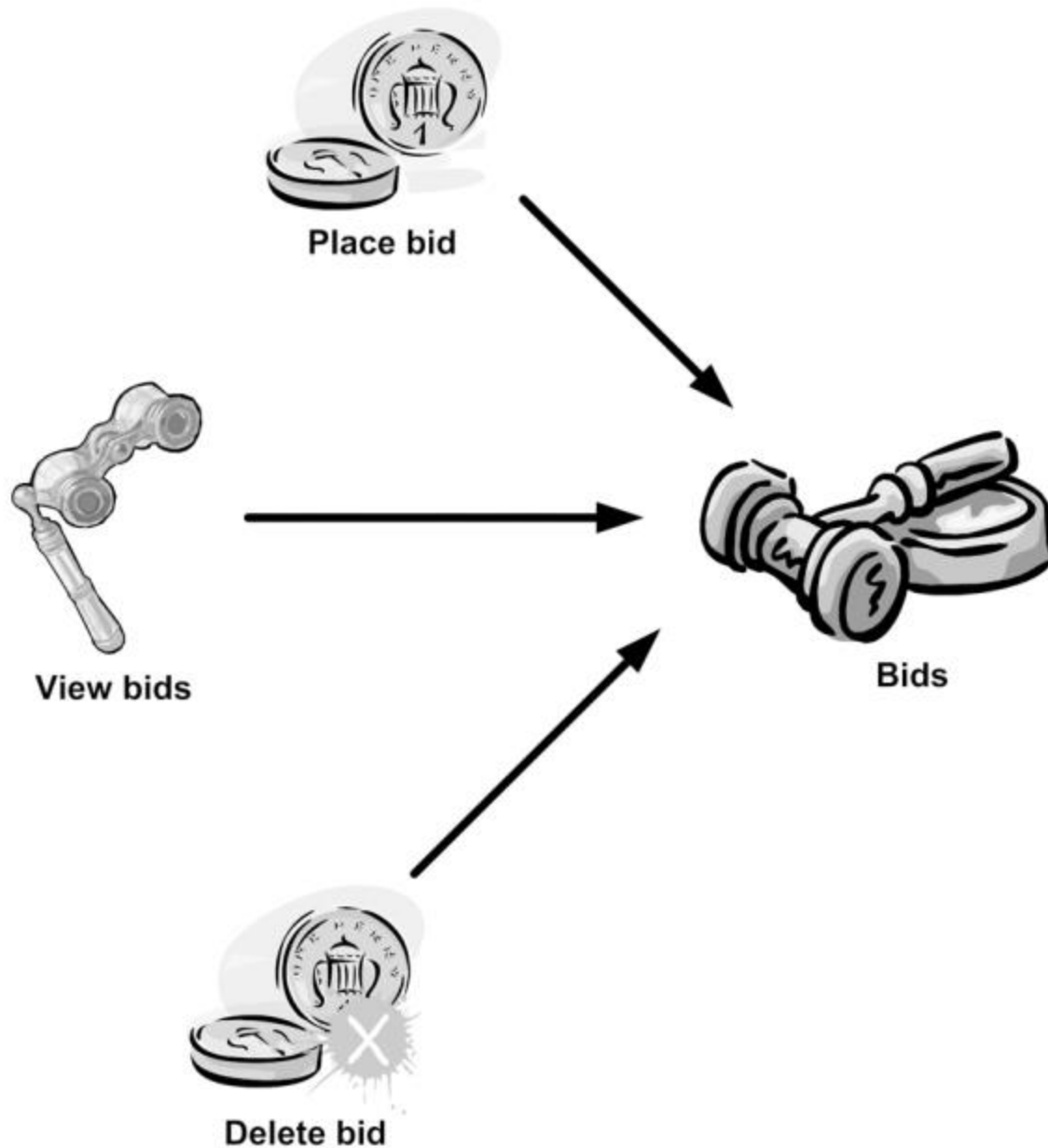13

Redhat Content – Chapter 3 Reviewing the Types of EJBs

Contents of **Chapter 3 from: RH student guide** removed for Copyright preservation.

Redhat Content – Chapter 3
Reviewing the Types of EJBs

Contents of **Chapter 3 from: RH student guide** removed for Copyright preservation.

Place bid

View bids

Bids

Delete bid

Stateless Session Beans Example

```java
@Stateless(name = "BidService")
public class DefaultBidService implements BidService {
  private Connection connection;

  @Resource(name = "jdbc/ActionBazaarDB")
  private DataSource dataSource;

  @PostConstruct
  public void initialize() {
    try {
      connection = dataSource.getConnection();
    } catch (SQLException sqle) {
      sqle.printStackTrace();
    }
  }

  public void addBid(Bid bid) {
    Long bidId = getBidId();
```

1 Marks as a stateless bean

2 Injects data source

3 Receives PostConstruct callback

17

Stateless Session Beans Example

```
try {
    Statement statement = connection.createStatement();
    statement.execute("INSERT INTO BIDS "
  + " (BID_ID, BIDDER, ITEM_ID, AMOUNT) VALUES( "
  + bidId
  + ", "
  + bid.getBidder().getUserId()
  + ", "
  + bid.getItem().getItemId()
  + ", "
  + bid.getBidPrice() + ")");
  } catch (Exception sqle) {
 sqle.printStackTrace();
  }
}

private Long getBidId() {
   ...Code for generating a unique key...
}
```

Redhat Content – Chapter 3 Reviewing the Types of EJBs
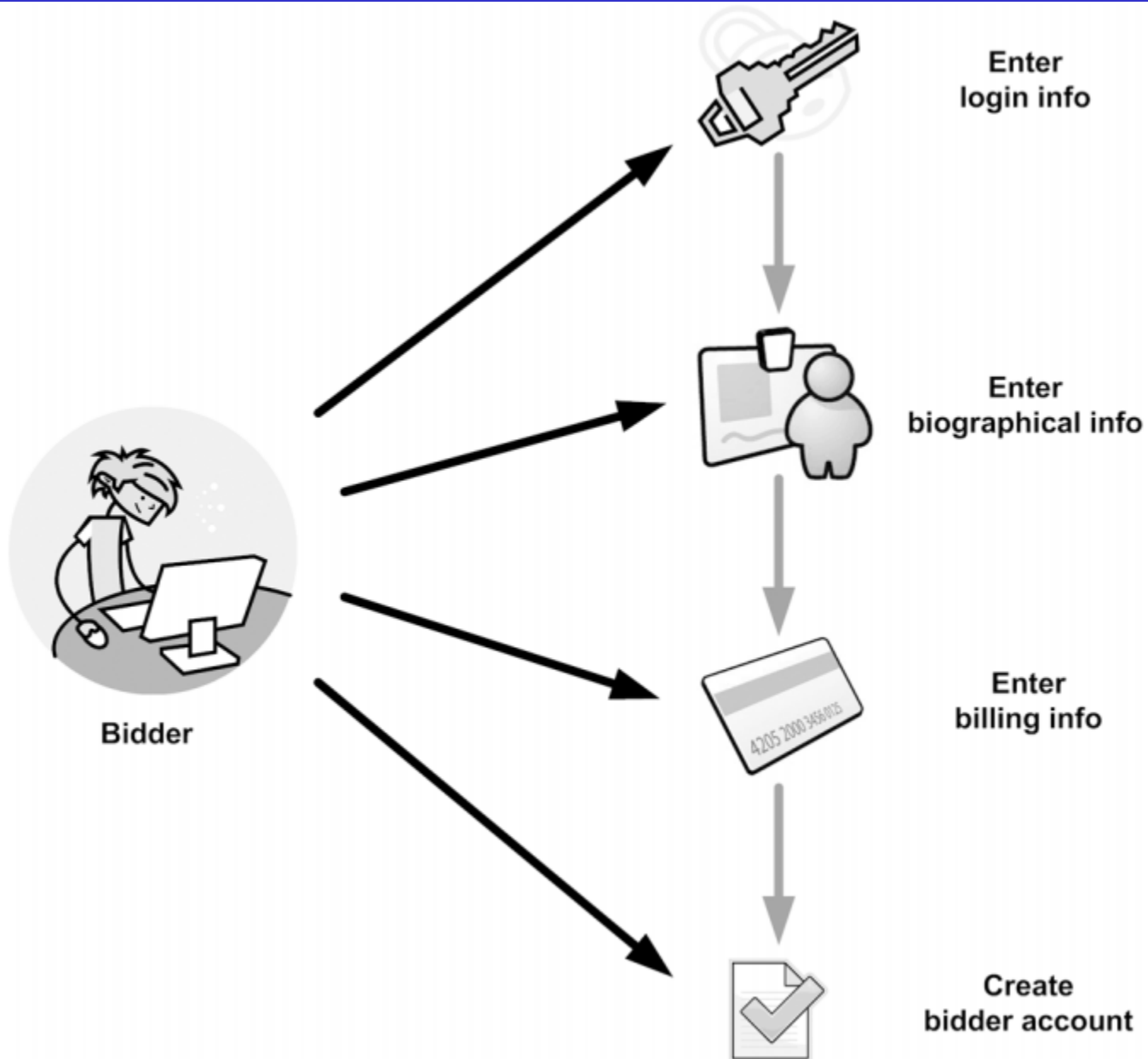
Contents of **Chapter 3 from: RH student guide** removed for Copyright preservation.

Redhat Content – Chapter 3 Reviewing the Types of EJBs

Contents of **Chapter 3 from: RH student guide** removed for Copyright preservation.

Enter login info

Enter biographical info

Enter billing info

Create bidder account

Bidder

21

Marks POJO stateful ❶

```java
@Stateful(name="BidderAccountCreator")
public class DefaultBidderAccountCreator implements BidderAccountCreator {
    @Resource(name = "jdbc/ActionBazaarDataSource")

    private DataSource dataSource;
    private Connection connection;
    private LoginInfo loginInfo;
    private BiographicalInfo biographicalInfo;
    private BillingInfo billingInfo;

    @PostConstruct
    @PostActivate
    public void openConnection() {
        try {
            connection = dataSource.getConnection();
        } catch (SQLException sqle) {
            sqle.printStackTrace();
        }
    }

    public void addLoginInfo(LoginInfo loginInfo) {
        this.loginInfo = loginInfo;
    }
```

❷ **Stateful instance variables**

❸ **Receives PostConstruct callback**

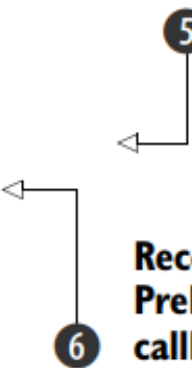❹ **Receives PostActivate callback**

22

```java
  public void addLoginInfo(LoginInfo loginInfo) {
    this.loginInfo = loginInfo;
  }

  public void addBiographicalInfo(BiographicalInfo biographicalInfo)
    throws WorkflowOrderViolationException {
    if (loginInfo == null) {
      throw new WorkflowOrderViolationException(
        "Login info must be set before biographical info");
    }
    this.biographicalInfo = biographicalInfo;
  }

  public void addBillingInfo(BillingInfo billingInfo)
    throws WorkflowOrderViolationException {
    if (biographicalInfo == null) {
      throw new WorkflowOrderViolationException(
        "Biographical info must be set before billing info");
    }
this.billingInfo = billingInfo;
  }

  @PrePassivate
  @PreDestroy
  public void cleanup() {
    try {
      connection.close();
      connection = null;
} catch (SQLException sqle) {
  sqle.printStackTrace();
}
```

⑤ Receives PrePassivate callback

⑥ Receives PreDestroy callback

Redhat Content – Chapter 3
Singleton Session Beans

Contents of **Chapter 3 from: RH student guide** removed for Copyright preservation.

When to Use Singleton Session Beans?

- You use singleton beans when you need to have a shared state that's application global

- Example:

- When an application starts up, there could be a need to check the database for consistency or verify that an external system is up and running.

  - Consider an application that is deployed on Glassfish

  - The application needs to connect to some server to function properly

  - Assume that the server is not guaranteed to start before the application

  - How could singleton beans server here?

When to Use Singleton Session Beans?

- Example: Action Bazaar featured deal
  - Each day ActionBazaar Company spotlights a particular item or featured deal.
  - Initially, the featured deal resided in the web tier.
  - However, currently ActionBazaar supports iPhone, Android apps that use a restful web service along with a dedicated mobile website
  - The featured deal logic is now pushed to the business logic tier.
    - Can we use stateful session beans?
    - Can we use stateless session beans?
    - Can we use singleton session beans?

Marks POJO as being a singleton session bean ❶

Listing 3.3   Singleton session bean example

Instantiates bean on startup ❷

Defines a bean dependency ❸

Code to be run immediately after creation ❹

Schedules featured item to be reloaded at midnight regularly ❺

```java
@Singleton
@Startup
@DependsOn("SystemInitializer")
public class DefaultFeaturedItem implements FeaturedItem {
    private Connection connection;

    @Resource(name = "jdbc/ActionBazaarDataSource")
    private DataSource dataSource;

    private Item featuredItem;

    @PostConstruct
    public void init() {
        try {
            connection = dataSource.getConnection();
        } catch (SQLException sqle) {
            sqle.printStackTrace();
        }

        loadFeaturedItem();
    }

    @Schedule(dayOfMonth="*",dayOfWeek="*",hour="0",minute="0",second="0")
    private void loadFeaturedItem() {
        featuredItem = ... load item from the database ...
    }

    @Override
    public Item getFeaturedItem() {
        return featuredItem;
    }
}
```

27

Message-Driven Beans

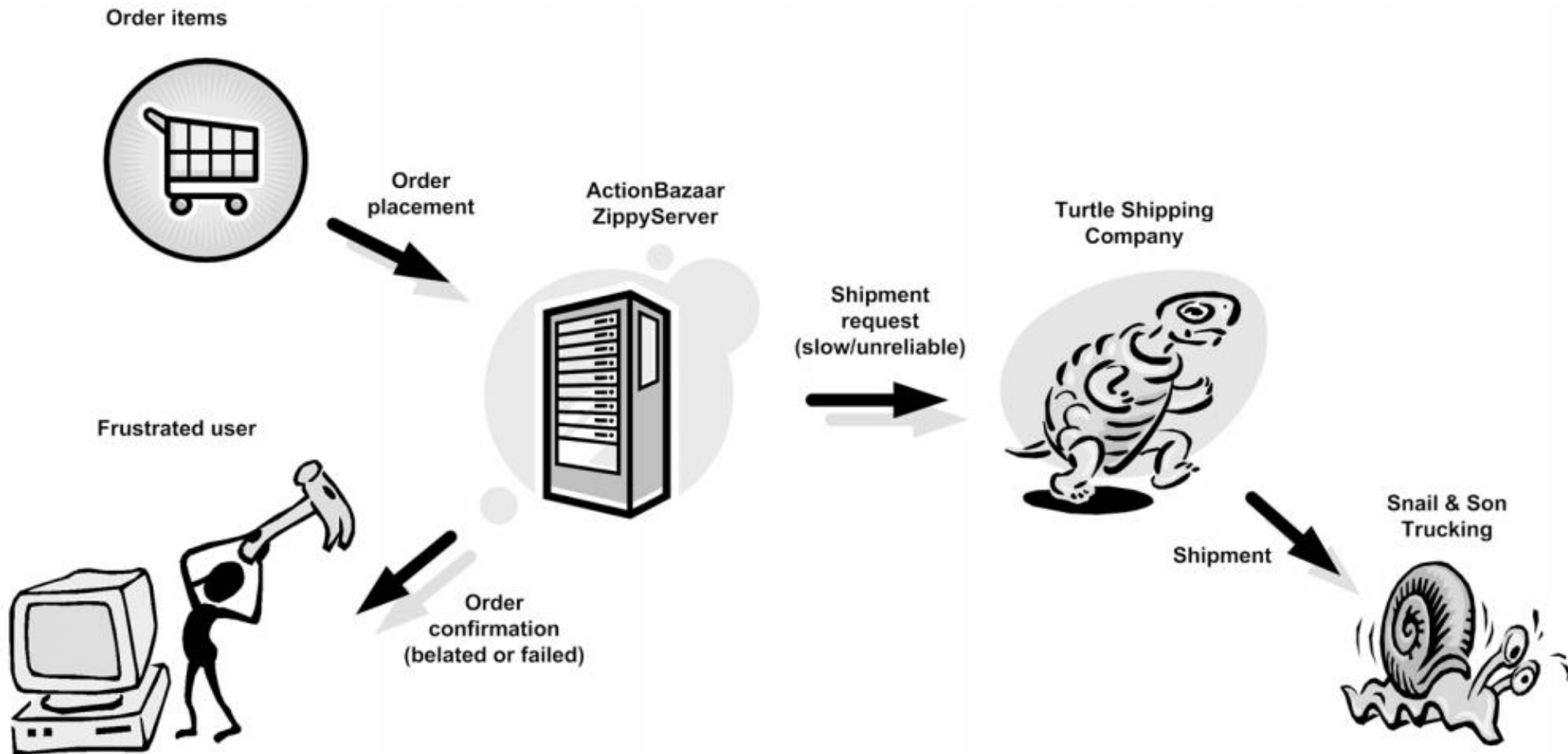Contents of **Chapter 3 from: RH student guide** removed for Copyright preservation.

Message-Driven Beans

Contents of **Chapter 3 from: RH student guide** removed for Copyright preservation.

Message-Driven Beans

- ActionBazaar is not a self contained, end-to-end system; its main business is in tracking auctions.

- Other business functions, like accounting and shipping, are handled by dedicated systems provided by specialized vendors.

- For example, ActionBazaar uses Turtle Shipping company to deliver items to winning bidders.

- When a bidder wins an auction, a shipment request is sent to Turtle's system in a synchronous fashion.

Message-Driven Beans

Message-Driven Beans
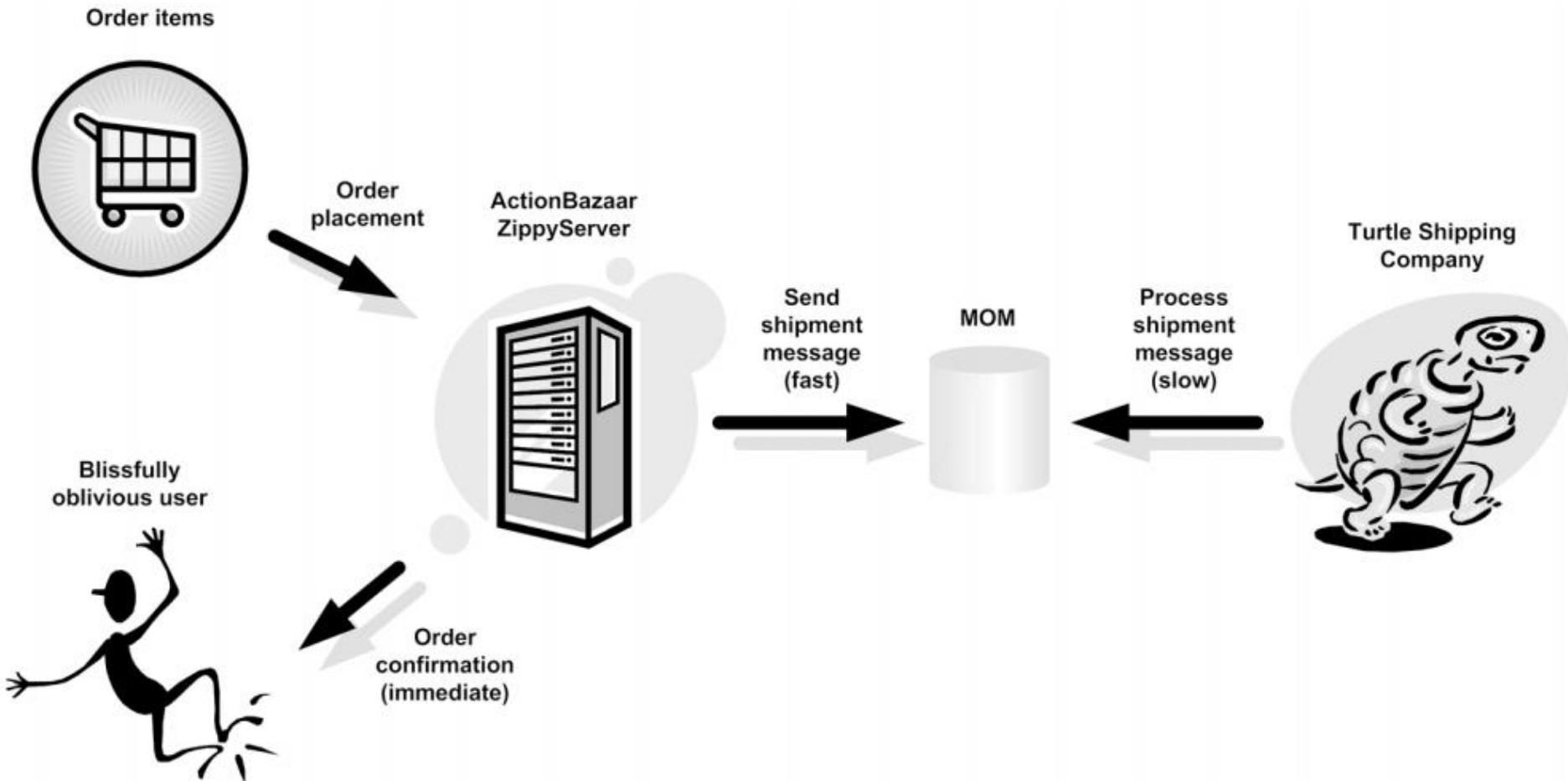
Describing the Life Cycle of a Stateful EJB

Contents of **Chapter 3 from: RH student guide** removed for Copyright preservation.

Describing the Life Cycle of a Stateless EJB

Contents of **Chapter 3 from: RH student guide** removed for Copyright preservation.

Describing the Life Cycle of a Singleton EJB

Contents of **Chapter 3 from: RH student guide** removed for Copyright preservation.

POJOs and Annotations

- Programming in EJB uses *Enterprise JavaBeanPOJOs* (plain old Java objects) together with Java *Enterprise JavaBean annotations*

- A bean is a plain old Java object: it consists of the application logic written simply in Java with no other code relating to it being a bean.

- Annotations are then used to ensure the correct behaviour in the EJB context.

POJOs and Annotations: Example

Example beans

- *@Stateful public class eShop implements Orders {...}*

- *@Stateless public class CalculatorBean implements Calculator {...}*

- *@MessageDriven public class SharePrice implements MessageListener {...}*

Example interfaces

- *@Remote public interface Orders {...}*

- *@Local public interface Calculator {...}*

Support for Distributed Systems' Features

- Beans are deployed to containers, and the containers provide implicit distributed system management using interception.

- The container provides the necessary policies in areas including transaction management, security, persistence and lifecycle management allowing the bean developer to focus exclusively on the application logic. (Recall Abstraction?)

- Let us see an example

Example: Transactions

- Transactions ensure that all components managed by a single server (or multiple servers in the case of distributed transactions) remain in a consistent state in spite of concurrent access from multiple clients and in the event of server failure.

- Example: In an eShop example, a transaction mechanism will ensure that:

  - two concurrent purchases do not result in a single item being sold twice

  - A server crash does not allow the system to get into an inconsistent state where an item has been paid for but not assigned to the purchaser

Example: Transactions

- The first thing to declare is whether transactions associated with an enterprise bean should be **bean-managed** or **container-managed** using an annotation:

    - *@TransactionManagement (BEAN)*
    - *@TransactionManagement (CONTAINER)*

Amazon

- Order for book (1 copy only)
- 2 clients need to buy the book

Example: Transactions

- *@TransactionManagement (BEAN)*

```
@Stateful
@TransactionManagement (BEAN)
public class eShop implements Orders {
    @Resource javax.transaction.UserTransaction ut;

    public void MakeOrder (...) {
        ut.begin ();
        ...
        ut.commit ();
    }
}
```

Example: Transactions

- *@TransactionManagement (Container)*

```
@Stateful public class eShop implements Orders {
    ...
    @TransactionAttribute (REQUIRED)
    public void MakeOrder (...) {
    ...
    }
}
```

43

Required Readings

- Panda, Debu, Reza Rahman, and Derek Lane. *EJB 3 in Action*. Vol. 15. Manning Publications Company, 2007.

- Chapter 8: **Distributed Systems**: **Concepts and Design**, 5th Edition. George **Coulouris**, Cambridge University. Jean Dollimore, Formerly of Queen Mary, University of London.

- Red Hat Application Development 1: Programming in Java EE Edition 2
  - Chapter 3: "Converting a POJO to an EJB" section
  - Chapter 3: "Describing the lifecycle of an EJB" section
  - Chapter 2: "Describing an application server" section