# Liferay Objects

Interview & Technical Reference Notes

---

**What are Liferay Objects?**

Liferay Objects provide a low-code/no-code mechanism for defining custom business entities and building CRUD-oriented applications without creating and deploying a traditional Service Builder module.
An administrator or developer can define the object's fields, relationships, validations, actions, layouts, permissions, and other behavior through Liferay's administration interfaces.
Liferay also generates APIs for interacting with object entries.

## Basic Concepts

### 1. What are Liferay Objects?

**Answer:** Liferay Objects are a low-code/no-code feature for creating custom business entities and CRUD applications without having to deploy traditional application code.

An Object Definition describes the structure and behavior of the business entity.

Typical configuration includes:

- Fields
- Relationships
- Layouts
- Views
- Validations
- Actions
- Permissions
- REST APIs

For example, an organization could create objects such as:

- Employee
- Student
- Course
- Purchase Request
- Asset Request
- Customer Feedback

### 2. What are the main Object scopes?

**Answer:** An object's scope determines where its entries belong and how they are separated within Liferay.

Common scopes include:

**Company Scope:** Entries belong to the Liferay virtual instance.

**Site Scope:** Entries are associated with individual Liferay Sites.

Site scope is particularly useful when the same Object Definition is used by multiple sites while each site maintains its own entries.

### 3. What are the core components of an Object?

**Answer:**

**Fields:** Define the data stored by an Object.

**Relationships:** Connect one Object Definition with another.

**Layouts:** Define how fields and relationships appear when users add or edit entries.

**Views:** Configure how object entries are presented in supported user interfaces.

**Actions:** Execute additional behavior when configured triggers occur.

**Validations:** Enforce business rules before data is accepted.

**Permissions:** Determine which users or roles can perform operations on objects and their entries.

### 4. What is the difference between System Objects and Custom Objects?

**Answer:**

**System Objects:** Object definitions associated with functionality provided and managed by Liferay. Their lifecycle is controlled by the platform.

**Custom Objects:** Business entities created by administrators or developers for application-specific requirements.

A custom object can represent entities such as:

$$Employee, \quad Student, \quad Vendor, \quad PurchaseRequest$$

## Intermediate Concepts

### 5. What relationship types are available?

**Answer:** Liferay Objects can model relationships between business entities.
Two important relationship patterns are:

- One-to-Many
- Many-to-Many

For example:

$$Department \longrightarrow \{Employee_1, Employee_2, \ldots\}$$

represents a one-to-many relationship.
A Student–Course relationship can naturally be modeled as many-to-many:

$$Student \longleftrightarrow Course$$

because one student may take several courses and one course may contain several students.

### 6. How do validations work?

**Answer:** Validations ensure that object entries satisfy defined business rules.
Basic field constraints may include requirements such as:

- Required values
- Valid field formats
- Appropriate numeric values
- Length constraints where supported

More sophisticated validation rules can enforce relationships between multiple fields.
For example, suppose an object contains:

$$P_o = OriginalPrice$$

and

$$P_d = DiscountPrice.$$

The required business constraint may be written as:

$$P_d \leq P_o.$$

Conceptually, a validation rule checks:

```
discountPrice <= originalPrice
```

If the condition fails, the entry should be rejected with an appropriate validation error.

### 7. What are Object Actions?

**Answer:** Object Actions allow additional business behavior to be executed when specific events occur.

Typical lifecycle events include operations around:

- Add
- Update
- Delete

Depending on the Liferay version and configuration, an action can be used to invoke mechanisms such as:

- Webhooks
- Notifications
- Supported scripting or custom execution mechanisms

A typical flow is:

$$ObjectEvent \rightarrow Action \rightarrow External/InternalOperation$$

For example:

$$EmployeeCreated \rightarrow Webhook \rightarrow HRIntegration$$

### 8. Can Liferay Workflow be used with Objects?

**Answer:** Yes. Object entries can participate in Liferay's workflow infrastructure when workflow is configured for the relevant object.

For example, a Purchase Request might follow:

$$Draft \rightarrow Submitted \rightarrow ManagerReview \rightarrow Approved/Rejected$$

This makes Objects useful for approval-driven applications without requiring a complete custom application to be developed from scratch.

### 9. What is a Picklist?

**Answer:** A Picklist provides a predefined set of values that can be reused by object fields.

For example:

```
Employment Status

Active
Inactive
On Leave
Terminated
```

A field can reference the Picklist instead of allowing arbitrary text input.

Conceptually:

$$ObjectField \rightarrow Picklist \rightarrow \{Option_1, Option_2, \ldots\}$$

This improves consistency and prevents invalid or misspelled values.

### 10. Do Objects provide REST APIs?

**Answer:** Yes. One of the major advantages of Liferay Objects is that REST APIs are generated for published custom objects.

A common endpoint pattern is:

```
/o/c/<pluralObjectName>
```

For an Employee object, an endpoint might resemble:

```
/o/c/employees
```

The API can support CRUD-oriented operations together with features such as filtering, sorting, and pagination where applicable.

For example:

```
GET /o/c/employees
POST /o/c/employees
GET /o/c/employees/{id}
PUT /o/c/employees/{id}
DELETE /o/c/employees/{id}
```

Access remains subject to Liferay authentication and authorization.

## Advanced Concepts

### 11. How does permissioning work?

**Answer:** Liferay Objects integrate with Liferay's role-based permission framework.

Permissions can control operations such as:

- Add
- View
- Update

- Delete

Entry-level permissions can further restrict access to individual object entries.
The overall authorization model can therefore be viewed as:

$$User \rightarrow Role \rightarrow ResourcePermission \rightarrow Object/ObjectEntry$$

This is important when building applications where different users must have different levels of access to the same business entity.

### 12. Can Objects be accessed from OSGi code?

**Answer:** Yes. Custom Java/OSGi modules can interact with Liferay Objects through Liferay's Object APIs and services.

A commonly used service is:

```
ObjectEntryLocalService
```

Conceptually, adding an entry involves supplying the object definition, user/context information, and a map containing the object's values.

For example:

```
Map<String, Serializable> values = new HashMap<>();

values.put("firstName", "John");
values.put("department", "Engineering");

// Use ObjectEntryLocalService with the parameters
// required by the target Liferay version.
```

Developers should use the exact `ObjectEntryLocalService` method signature available in their installed Liferay version, because service signatures can vary between releases.

Model listeners and related extension mechanisms can also be used when custom Java behavior needs to respond to persistence lifecycle events.

### 13. Can Object entries be imported and exported?

**Answer:** Liferay provides import/export capabilities for supported Object entry use cases.

This is particularly useful for:

- Initial data migration
- Bulk entry creation
- Data exchange
- Administrative maintenance
- Testing

CSV-based operations are especially useful for business users managing large numbers of entries.

### 14. What is an Object Layout?

**Answer:** An Object Layout controls the organization and presentation of fields and relationships on an Object entry form.

For example, an Employee object could organize fields as:

```
Personal Information
    First Name
    Last Name
    Email

Employment Information
```

```
    Employee ID
    Department
    Joining Date

Compensation
    Salary
    Bonus
```

Layouts improve usability by grouping related information and reducing the complexity of large forms.

Sensitive-field visibility should ultimately be enforced through the appropriate Liferay permissions and application design rather than relying only on visual layout differences.

### 15. What are the limitations of Liferay Objects?

**Answer:** Objects are highly effective for low-code business applications, but they are not intended to replace every custom persistence requirement.

Potential considerations include:

- Highly complex querying may require custom development.
- Advanced cross-entity reporting may require external analytics or custom APIs.
- Direct SQL access should not be treated as the normal Object programming model.
- Very large datasets require appropriate performance testing, indexing strategy, API design, and infrastructure sizing.
- Complex transactional business logic may be better implemented in Java services.
- Specialized persistence requirements may be better suited to Service Builder or another dedicated backend.

There is no universal record-count threshold at which Objects suddenly become unsuitable. Performance depends on query patterns, indexing, relationships, infrastructure, Liferay version, and application design.

## Objects vs. Service Builder

### 16. When should Objects be used instead of Service Builder?

| **Feature** | Liferay Objects | Service Builder |
|---|---|---|
| **Development Style** | Low-code / configuration driven | Java / OSGi development |
| **CRUD** | Generated automatically | Generated from service.xml with developer customization |
| **REST APIs** | Generated for published objects | Typically implemented/exposed through Liferay's service/API mechanisms |
| **Business Logic** | Good for relatively straightforward business processes | Suitable for sophisticated custom business logic |
| **Custom Queries** | More constrained | Supports custom finders and persistence customization |
| **Deployment** | Many changes can be configured through Liferay | Requires building and deploying modules |
| **Typical Use** | Requests, directories, forms, workflows, administrative applications | Complex enterprise applications and custom domain services |

A useful architectural rule is:

$$Configuration - DrivenCRUD \Rightarrow LiferayObjects$$

whereas:

$$ComplexDomainLogic + SpecializedPersistence \Rightarrow ServiceBuilder$$

Objects and Service Builder can also coexist in the same solution.

## Real-World Example: Student–Course Application

### 17. How can a Student–Course application be created using Objects?

Consider a university portal where students can register for courses.

#### Step 1: Create the Student Object

Create an Object Definition:

```
Student
```

Example fields:

| Field | Type |
|---|---|
| studentId | Text |
| firstName | Text |
| lastName | Text |
| email | Text |
| dateOfBirth | Date |
| status | Picklist |

### Step 2: Create the Course Object

Create another Object Definition:

```
Course
```

Example fields:

| Field | Type |
|---|---|
| courseCode | Text |
| courseName | Text |
| credits | Integer |
| department | Text/Picklist |

### Step 3: Create the Relationship

A student may register for several courses, and a course may contain many students.

Therefore:

$$StudentM : N \longleftrightarrow Course$$

A many-to-many Object relationship can represent this association.

### Step 4: Configure Layouts

Create logical sections for Student entries:

```
Student Information

Personal Details
Academic Details
Course Registrations
```

This provides administrators with a cleaner entry-management experience.

### Step 5: Manage Related Entries

Configure the user interface so that administrators or authorized users can associate Course entries with Student entries.

Conceptually:

$$StudentEntry \rightarrow Courses \rightarrow AddRelatedEntry$$

### Step 6: Add Automation

When a new student is created, an Object Action could trigger a notification or invoke an external service.

For example:

$$StudentCreated \rightarrow ObjectAction \rightarrow WelcomeNotification$$

A webhook could instead invoke an external service:

$$LiferayObject \rightarrow Webhook \rightarrow ExternalStudentService$$

### Step 7: Access Through REST

Once the objects are published, their generated APIs can be consumed by external applications.

Conceptually:

```
GET /o/c/students
POST /o/c/students

GET /o/c/courses
POST /o/c/courses
```

This makes it possible to connect:

$$React/LiferayUI \rightarrow LiferayObjectRESTAPI \rightarrow ObjectEntries$$

## Architecture Summary

A typical Object-based application can be visualized as:

$$\begin{array}{c} LiferayUI/React/ExternalApplication \\ \downarrow \\ RESTAPI - - - /o/c/ < object > \\ \downarrow \\ ObjectDefinition \\ \downarrow \\ ObjectEntries \\ \downarrow \\ LiferayPersistenceLayer \end{array}$$

The Object Definition additionally controls:

$$\begin{cases} Fields \\ Relationships \\ Validations \\ Actions \\ Layouts \\ Permissions \\ Workflow \end{cases}$$

## Interview Takeaway

**Key Point**

Liferay Objects are best understood as a configurable business-object layer inside Liferay. They provide data modeling, CRUD operations, relationships, permissions, APIs, validations, workflows, layouts, and automation with significantly less custom Java development than a traditional Service Builder application.

Use **Objects** when the requirement is primarily configuration-driven business data and CRUD functionality.

Use **Service Builder/custom OSGi services** when the application requires sophisticated persistence, complex transactional logic, specialized queries, or deeper Java-level control.