Integrating Apache Cassandra with a Java/Spring Boot application using the Abstract Factory, JPA, and Repository patterns involves a few steps. Cassandra is generally used with the DataStax Java Driver rather than JPA because JPA is typically associated with relational databases. However, you can use Spring Data Cassandra for a similar abstraction layer.

Here's how you can achieve this integration step-by-step:

1. Setup Maven Dependencies

Add the necessary dependencies for Spring Boot and Spring Data Cassandra to your `pom.xml`:

```
<dependencies>
    <!-- Spring Boot Starter for Cassandra -->
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-data-cassandra</artifactId>
    </dependency>

    <!-- Spring Boot Starter for Data JPA -->
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-data-jpa</artifactId>
    </dependency>

    <!-- DataStax Java Driver for Cassandra -->
    <dependency>
        <groupId>com.datastax.oss</groupId>
        <artifactId>java-driver-core</artifactId>
        <version>4.14.0</version> <!-- Check for the latest version -->
    </dependency>

    <!-- Spring Boot Starter for Web (Optional) -->
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
    </dependency>
</dependencies>
```

2. Configure Cassandra

In `application.properties` or `application.yml`, configure your Cassandra connection settings:

```
# application.properties
spring.data.cassandra.keyspace-name=my_keyspace
spring.data.cassandra.contact-points=localhost:9042
spring.data.cassandra.local-datacenter=datacenter1
```

3. Define the Entity and Repository

Create a Cassandra entity and a repository interface.

Entity Class

```
import org.springframework.data.cassandra.core.mapping.PrimaryKey;
import org.springframework.data.cassandra.core.mapping.Table;

@Table("my_table")
public class MyEntity {

    @PrimaryKey
    private String partitionKey;
    private String sortKey;
    private String someOtherField;

    // Getters and setters
    public String getPartitionKey() {
        return partitionKey;
    }

    public void setPartitionKey(String partitionKey) {
        this.partitionKey = partitionKey;
    }

    public String getSortKey() {
        return sortKey;
    }

    public void setSortKey(String sortKey) {
        this.sortKey = sortKey;
    }

    public String getSomeOtherField() {
        return someOtherField;
    }

    public void setSomeOtherField(String someOtherField) {
        this.someOtherField = someOtherField;
    }
}
```

Repository Interface

```
import org.springframework.data.cassandra.repository.CassandraRepository;
import org.springframework.stereotype.Repository;

import java.util.List;

@Repository
public interface MyEntityRepository extends CassandraRepository<MyEntity, String> {

    List<MyEntity> findByPartitionKeyAndSortKeyStartingWith(String partitionKey, String sortKeyP
}
```

4. Implement the Abstract Factory Pattern

Define an abstract factory interface and its implementation for creating repositories.

Abstract Factory Interface

```
public interface RepositoryFactory {
    MyEntityRepository createMyEntityRepository();
}
```

Factory Implementation

```
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Component;

@Component
public class RepositoryFactoryImpl implements RepositoryFactory {

    private final MyEntityRepository myEntityRepository;

    @Autowired
    public RepositoryFactoryImpl(MyEntityRepository myEntityRepository) {
        this.myEntityRepository = myEntityRepository;
    }

    @Override
    public MyEntityRepository createMyEntityRepository() {
        return myEntityRepository;
    }
}
```

5. Use the Repository in a Service

Create a service class that uses the repository via the factory.

Service Class

```
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import java.util.List;

@Service
public class MyEntityService {

    private final RepositoryFactory repositoryFactory;

    @Autowired
    public MyEntityService(RepositoryFactory repositoryFactory) {
        this.repositoryFactory = repositoryFactory;
    }

    public List<MyEntity> findEntities(String partitionKey, String sortKeyPrefix) {
        MyEntityRepository repository = repositoryFactory.createMyEntityRepository();
        return repository.findByPartitionKeyAndSortKeyStartingWith(partitionKey, sortKeyPrefix);
    }
}
```

6. Create a Controller (Optional)

If you are building a web application, you might want to expose an API endpoint.

Controller Class

```
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.RestController;

import java.util.List;

@RestController
public class MyEntityController {

    private final MyEntityService myEntityService;

    @Autowired
    public MyEntityController(MyEntityService myEntityService) {
        this.myEntityService = myEntityService;
    }

    @GetMapping("/entities")
    public List<MyEntity> getEntities(
            @RequestParam String partitionKey,
            @RequestParam String sortKeyPrefix) {
        return myEntityService.findEntities(partitionKey, sortKeyPrefix);
    }
}
```

Summary

1. **Maven Dependencies**: Add necessary dependencies for Spring Boot and Cassandra.
2. **Configuration**: Set up Cassandra connection properties.
3. **Entity and Repository**: Define the entity and repository interface for Cassandra.
4. **Abstract Factory**: Implement the abstract factory pattern for creating repositories.
5. **Service**: Create a service to use the repository.
6. **Controller**: Optionally, create a REST controller to expose an API.

This setup integrates Cassandra with a Spring Boot application using the Abstract Factory and Repository patterns, providing a clean architecture for managing Cassandra interactions.