Glarimy Technology Services
www.glarimy.com

Clean Code Java

Krishna Mohan Koyya
https://www.linkedin.com/in/krishnamohankoyya
https://www.opensourceforu.com/author/krishna-mohan-koyya
krishna@glarimy.com | 9731423166

User Directory

- User Directory is a standalone application to maintain a database of users.
- The default implementation stores the user data in memory.
  - However, the customers should be able to plug in any kind of storage technology.
- The user data include a unique phone number, name, email and optionally address.
- It should be possible to add new users and find users either by their phone number or name.
- It should be possible to log all activities on the User Directory.
- It should be possible to collect response time metrics on all activities on the User Directory.

Glarimy Technology Services
www.glarimy.com

Clean Code Java

Java Beans

Java Beans

```java
public class Address {
   private String location;
   private String city;
   private int pincode;

   public String getLocation() {
      return location;
   }

   public void setLocation(String location) {
      this.location = location;
   }

   public String getCity() {
      return city;
   }

   public void setCity(String city) {
      this.city = city;
   }

   public int getPincode() {
      return pincode;
   }

   public void setPincode(int pincode) {
      this.pincode = pincode;
   }
}
```

Java Beans

```java
public class User {
   private long phone;
   private String name;
   private String email;
   private Address address;

   public long getPhone() {
      return phone;
   }

   public void setPhone(long phone) {
      this.phone = phone;
   }

   public String getName() {
      return name;
   }

   public void setName(String name) {
      this.name = name;
   }

   // other getters and setters

}
```

Basic Service

```
public class SimpleDirectory {
   private Map<Long, User> users = new HashMap<Long, User>();

   public User add(User user) {
      users.put(user.getPhone(), user);
      return user;
   }

   public User findByPhone(long phone) {
      return users.get(phone);
   }

   public List<User> searchByName(String name) {
      List<User> results = new ArrayList<User>();
      for (User user : users.values()) {
         if (user.getName().contains(name))
            results.add(user);
      }
      return results;
   }

}
```

Client

```java
public class DirectoryConsole {
   public static void main(String[] args) {
      SimpleDirectory dir = new SimpleDirectory();
      User user = new User();
      user.setPhone(123L);
      user.setName("Krishna");
      user.setEmail("krishna@glarimy.com");

      dir.add(user);
      User result = dir.findByPhone(123L);
      System.out.println("Email: " + result.getEmail());
      List<User> results = dir.searchByName("Krishna");
      System.out.println("Number of results: " + results.size());
   }
}
```

Client

```
public class DirectoryConsole {
   public static void main(String[] args) {
      SimpleDirectory dir = new SimpleDirectory();
      User user = new User();
      user.setPhone(123L);
      user.setName("Krishna");
      user.setEmail("krishna@glarimy.com");

      dir.add(user);
      User result = dir.findByPhone(123L);
      System.out.println("Email: " + result.getEmail());
      List<User> results = dir.searchByName("Krishna");
      System.out.println("Number of results: " + results.size());
   }
}
```

Glarimy Technology Services
www.glarimy.com

Clean Code Java

Interfaces

Service interface

```
public interface Directory {
   public void add(User user);

   public User findByPhone(long phone);

   public List<User> searchByName(String name);
}
```

Service interface

```
public interface Directory {
   public boolean add(User user);

   public User findByPhone(long phone);

   public List<User> searchByName(String name);
}
```

Service interface

```
public interface Directory {
   public User add(User user);

   public User findByPhone(long phone);

   public List<User> searchByName(String name);
}
```

Service Implementation

```
public class SimpleDirectory implements Directory {
   private Map<Long, User> users = new HashMap<Long, User>();

   public User add(User user) {
      users.put(user.getPhone(), user);
      return user;
   }

   public User findByPhone(long phone) {
      return users.get(phone);
   }

   public List<User> searchByName(String name) {
      List<User> results = new ArrayList<User>();
      for (User user : users.values()) {
         if (user.getName().contains(name))
            results.add(user);
      }
      return results;
   }

}
```

Service Implementation

```
public class SimpleDirectory implements Directory {
   private Map<Long, User> users = new HashMap<Long, User>();

   public User add(User user) {
      users.put(user.getPhone(), user);
      return user;
   }

   public User findByPhone(long phone) {
      return users.get(phone);
   }

   public List<User> searchByName(String name) {
      List<User> results = new ArrayList<User>();
      for (User user : users.values()) {
         if (user.getName().contains(name))
            results.add(user);
      }
      return results;
   }

}
```

Client

```java
public class DirectoryConsole {
   public static void main(String[] args) {
      Directory dir = new SimpleDirectory();
      User user = new User();
      user.setPhone(123L);
      user.setName("Krishna");
      user.setEmail("krishna@glarimy.com");

      dir.add(user);
      User result = dir.findByPhone(123L);
      System.out.println("Email: " + result.getEmail());
      List<User> results = dir.searchByName("Krishna");
      System.out.println("Number of results: " + results.size());
   }
}
```

Client

```java
public class DirectoryConsole {
   public static void main(String[] args) {
      Directory dir = new SimpleDirectory();
      User user = new User();
      user.setPhone(123L);
      user.setName("Krishna");
      user.setEmail("krishna@glarimy.com");

      dir.add(user);
      User result = dir.findByPhone(123L);
      System.out.println("Email: " + result.getEmail());
      List<User> results = dir.searchByName("Krishna");
      System.out.println("Number of results: " + results.size());
   }
}
```

Glarimy Technology Services
www.glarimy.com

Clean Code Java

Exceptions

Robust service interface

```
public interface Directory {
   public User add(User user) throws Exception;

   public User findByPhone(long phone) throws Exception;

   public List<User> searchByName(String name);
}
```

Robust service interface

```
public interface Directory {
   public User add(User user) throws Exception;

   public User findByPhone(long phone) throws Exception;

   public List<User> searchByName(String name);
}
```

Robust service interface

```
public interface Directory {
   public User add(User user) throws Exception;

   public User findByPhone(long phone) throws Exception;

   public List<User> searchByName(String name);
}
```

Custom exception

```java
@SuppressWarnings("serial")
public class InvalidUserException extends Exception {

   public InvalidUserException() {

   }

   public InvalidUserException(String message) {
      super(message);
   }

   public InvalidUserException(Throwable cause) {
      super(cause);
   }

   public InvalidUserException(String message, Throwable cause) {
      super(message, cause);
   }

   public InvalidUserException(String message, Throwable cause, boolean enableSuppression,
         boolean writableStackTrace) {
      super(message, cause, enableSuppression, writableStackTrace);
   }

}
```

Custom exception

```java
@SuppressWarnings("serial")
public class InvalidUserException extends Exception {

   public InvalidUserException() {

   }

   public InvalidUserException(String message) {
      super(message);
   }

   public InvalidUserException(Throwable cause) {
      super(cause);
   }

   public InvalidUserException(String message, Throwable cause) {
      super(message, cause);
   }

   public InvalidUserException(String message, Throwable cause, boolean enableSuppression,
         boolean writableStackTrace) {
      super(message, cause, enableSuppression, writableStackTrace);
   }

}
```

Custom exception

```java
@SuppressWarnings("serial")
public class DuplicateUserException extends Exception {

   public DuplicateUserException() {

   }

   public DuplicateUserException(String message) {
      super(message);
   }

   public DuplicateUserException(Throwable cause) {
      super(cause);
   }

   public DuplicateUserException(String message, Throwable cause) {
      super(message, cause);
   }

   public DuplicateUserException(String message, Throwable cause, boolean enableSuppression,
         boolean writableStackTrace) {
      super(message, cause, enableSuppression, writableStackTrace);
   }

}
```

Custom exception

```java
@SuppressWarnings("serial")
public class UserNotFoundException extends Exception {

   public UserNotFoundException() {

   }

   public UserNotFoundException(String arg0) {
      super(arg0);
   }

   public UserNotFoundException(Throwable arg0) {
      super(arg0);
   }

   public UserNotFoundException(String arg0, Throwable arg1) {
      super(arg0, arg1);
   }

   public UserNotFoundException(String arg0, Throwable arg1, boolean arg2, boolean arg3) {
      super(arg0, arg1, arg2, arg3);
   }

}
```

Robust service interface

```
public interface Directory {
   public User add(User user) throws InvalidUserException, DuplicateUserException;

   public User findByPhone(long phone) throws UserNotFoundException;

   public List<User> searchByName(String name);
}
```

Robust service interface

```
public interface Directory {
   public User add(User user) throws InvalidUserException, DuplicateUserException;

   public User findByPhone(long phone) throws UserNotFoundException;

   public List<User> searchByName(String name);
}
```

Robust service interface

```
public interface Directory {
   public User add(User user) throws InvalidUserException, DuplicateUserException;

   public User findByPhone(long phone) throws UserNotFoundException;

   public List<User> searchByName(String name);
}
```

Robust service implementation

```
public class SimpleDirectory implements Directory {
   private Map<Long, User> users = new HashMap<Long, User>();

   public User add(User user) throws InvalidUserException, DuplicateUserException{
      if (user == null)
         throw new DirectoryException("Invalid User");
      if (user.getName() == null || user.getName().trim().length() < 6)
         throw new InvalidUserException("Invalid name");
      if (user.getPhone() < 0 && Long.toString(user.getPhone()).length() != 10)
         throw new InvalidUserException("Invalid phone number");
      if (user.getEmail() == null || !user.getEmail().contains("@"))
         throw new InvalidUserException("Invalid email");

      if (users.containsKey(user.getPhone()))
         throw new DuplicateUserException("User already exists");

      users.put(user.getPhone(), user);
      return user;
   }

   // TODO: Other methods

}
```

Robust service implementation

```
public class SimpleDirectory implements Directory {
   private Map<Long, User> users = new HashMap<Long, User>();

   public User add(User user) throws InvalidUserException, DuplicateUserException {
      if (user == null)
         throw new DirectoryException("Invalid User");
      if (user.getName() == null || user.getName().trim().length() < 6)
         throw new InvalidUserException("Invalid name");
      if (user.getPhone() < 0 && Long.toString(user.getPhone()).length() != 10)
         throw new InvalidUserException("Invalid phone number");
      if (user.getEmail() == null || !user.getEmail().contains("@"))
         throw new InvalidUserException("Invalid email");

      if (users.containsKey(user.getPhone()))
         throw new DuplicateUserException("User already exists");

      users.put(user.getPhone(), user);
      return user;
   }

   // TODO: Other methods

}
```

Robust service implementation

```java
public class SimpleDirectory implements Directory {
   private Map<Long, User> users = new HashMap<Long, User>();

   public User add(User user) throws InvalidUserException, DuplicateUserException{
      if (user == null)
         throw new DirectoryException("Invalid User");
      if (user.getName() == null || user.getName().trim().length() < 6)
         throw new InvalidUserException("Invalid name");
      if (user.getPhone() < 0 && Long.toString(user.getPhone()).length() != 10)
         throw new InvalidUserException("Invalid phone number");
      if (user.getEmail() == null || !user.getEmail().contains("@"))
         throw new InvalidUserException("Invalid email");

      if (users.containsKey(user.getPhone()))
         throw new DuplicateUserException("User already exists");

      users.put(user.getPhone(), user);
      return user;
   }

   // TODO: Other methods

}
```

Robust service implementation

```
public class SimpleDirectory implements Directory {
   private Map<Long, User> users = new HashMap<Long, User>();

   public User add(User user) throws InvalidUserException, DuplicateUserException{
      if (user == null)
         throw new DirectoryException("Invalid User");
      if (user.getName() == null || user.getName().trim().length() < 6)
         throw new InvalidUserException("Invalid name");
      if (user.getPhone() < 0 && Long.toString(user.getPhone()).length() != 10)
         throw new InvalidUserException("Invalid phone number");
      if (user.getEmail() == null || !user.getEmail().contains("@"))
         throw new InvalidUserException("Invalid email");

      if (users.containsKey(user.getPhone()))
         throw new DuplicateUserException("User already exists");

      users.put(user.getPhone(), user);
      return user;
   }

   // TODO: Other methods

}
```

Robust service implementation

```
public class SimpleDirectory implements Directory {
   private Map<Long, User> users = new HashMap<Long, User>();

   // other methods

   public User findByPhone(long phone) throws UserNotFoundException{
      User result = users.get(phone);
      if (result == null)
         throw new UserNotFoundException();
      return result;
   }

   public List<User> searchByName(String name) {
      List<User> results = new ArrayList<User>();
      for (User user : users.values()) {
         if (user.getName().contains(name))
            results.add(user);
      }
      return results;
   }
}
```

Robust service implementation

```
public class SimpleDirectory implements Directory {
   private Map<Long, User> users = new HashMap<Long, User>();

   // other methods

   public User findByPhone(long phone) throws UserNotFoundException{
      User result = users.get(phone);
      if (result == null)
         throw new UserNotFoundException();
      return result;
   }

   public List<User> searchByName(String name) {
      List<User> results = new ArrayList<User>();
      for (User user : users.values()) {
         if (user.getName().contains(name))
            results.add(user);
      }
      return results;
   }
}
```

Robust service implementation

```
public class SimpleDirectory implements Directory {
   private Map<Long, User> users = new HashMap<Long, User>();

   // other methods

   public User findByPhone(long phone) throws UserNotFoundException{
      User result = users.get(phone);
      if (result == null)
         throw new UserNotFoundException();
      return result;
   }

   public List<User> searchByName(String name) {
      List<User> results = new ArrayList<User>();
      for (User user : users.values()) {
         if (user.getName().contains(name))
            results.add(user);
      }
      return results;
   }
}
```

Generic Exception

```
@SuppressWarnings("serial")
public class DirectoryException extends Exception {

   public DirectoryException() {

   }

   public DirectoryException(String arg0) {
      super(arg0);
   }

   public DirectoryException(Throwable arg0) {
      super(arg0);
   }

   public DirectoryException(String arg0, Throwable arg1) {
      super(arg0, arg1);
   }

   public DirectoryException(String arg0, Throwable arg1, boolean arg2, boolean arg3) {
      super(arg0, arg1, arg2, arg3);
   }

}
```

Robust service interface

```
public interface Directory {
   public User add(User user) throws InvalidUserException, DuplicateUserException, DirectoryException;

   public User findByPhone(long phone) throws UserNotFoundException, DirectoryException;

   public List<User> searchByName(String name) throws DirectoryException;
}
```

Robust service interface

```
public interface Directory {
   public User add(User user) throws InvalidUserException, DuplicateUserException, DirectoryException;

   public User findByPhone(long phone) throws UserNotFoundException, DirectoryException;

   public List<User> searchByName(String name) throws DirectoryException;
}
```

Robust service interface

```
public interface Directory {
   public User add(User user) throws InvalidUserException, DuplicateUserException, DirectoryException;

   public User findByPhone(long phone) throws UserNotFoundException, DirectoryException;

   public List<User> searchByName(String name) throws DirectoryException;
}
```

Robust service interface

```
public interface Directory {
   public User add(User user) throws InvalidUserException, DuplicateUserException, DirectoryException;

   public User findByPhone(long phone) throws UserNotFoundException, DirectoryException;

   public List<User> searchByName(String name) throws DirectoryException;
}
```

Robust service implementation

```java
public class SimpleDirectory implements Directory {
   private Map<Long, User> users = new HashMap<Long, User>();

   public User add(User user) throws InvalidUserException, DuplicateUserException, DirectoryException {
      if (user == null)
         throw new DirectoryException("Invalid User");
      if (user.getName() == null || user.getName().trim().length() < 6)
         throw new InvalidUserException("Invalid name");
      if (user.getPhone() < 0 && Long.toString(user.getPhone()).length() != 10)
         throw new InvalidUserException("Invalid phone number");
      if (user.getEmail() == null || !user.getEmail().contains("@"))
         throw new InvalidUserException("Invalid email");

      if (users.containsKey(user.getPhone()))
         throw new DuplicateUserException("User already exists");

      users.put(user.getPhone(), user);
      return user;
   }

   // TODO: Other methods

}
```

Robust service implementation

```
public class SimpleDirectory implements Directory {
   private Map<Long, User> users = new HashMap<Long, User>();

   // other methods

   public User findByPhone(long phone) throws UserNotFoundException, DirectoryException {
      User result = users.get(phone);
      if (result == null)
         throw new UserNotFoundException();
      return result;
   }

   public List<User> searchByName(String name) throws DirectoryException {
      List<User> results = new ArrayList<User>();
      for (User user : users.values()) {
         if (user.getName().contains(name))
            results.add(user);
      }
      return results;
   }
}
```

Exception Hierarchy

```java
@SuppressWarnings("serial")
public class InvalidUserException extends DirectoryException {

   public InvalidUserException() {

   }

   public InvalidUserException(String message) {
      super(message);
   }

   public InvalidUserException(Throwable cause) {
      super(cause);
   }

   public InvalidUserException(String message, Throwable cause) {
      super(message, cause);
   }

   public InvalidUserException(String message, Throwable cause, boolean enableSuppression,
         boolean writableStackTrace) {
      super(message, cause, enableSuppression, writableStackTrace);
   }

}
```

Exception Hierarchy

```java
@SuppressWarnings("serial")
public class InvalidUserException extends DirectoryException {

   public InvalidUserException() {

   }

   public InvalidUserException(String message) {
      super(message);
   }

   public InvalidUserException(Throwable cause) {
      super(cause);
   }

   public InvalidUserException(String message, Throwable cause) {
      super(message, cause);
   }

   public InvalidUserException(String message, Throwable cause, boolean enableSuppression,
         boolean writableStackTrace) {
      super(message, cause, enableSuppression, writableStackTrace);
   }

}
```

Exception Hierarchy

```java
@SuppressWarnings("serial")
public class InvalidUserException extends DirectoryException {

   public InvalidUserException() {

   }

   public InvalidUserException(String message) {
      super(message);
   }

   public InvalidUserException(Throwable cause) {
      super(cause);
   }

   public InvalidUserException(String message, Throwable cause) {
      super(message, cause);
   }

   public InvalidUserException(String message, Throwable cause, boolean enableSuppression,
         boolean writableStackTrace) {
      super(message, cause, enableSuppression, writableStackTrace);
   }

}
```

Exception Hierarchy

```java
@SuppressWarnings("serial")
public class DuplicateUserException extends DirectoryException {

   public DuplicateUserException() {

   }

   public DuplicateUserException(String message) {
      super(message);
   }

   public DuplicateUserException(Throwable cause) {
      super(cause);
   }

   public DuplicateUserException(String message, Throwable cause) {
      super(message, cause);
   }

   public DuplicateUserException(String message, Throwable cause, boolean enableSuppression,
         boolean writableStackTrace) {
      super(message, cause, enableSuppression, writableStackTrace);
   }

}
```

Exception Hierarchy

```java
@SuppressWarnings("serial")
public class UserNotFoundException extends DirectoryException {

   public UserNotFoundException() {

   }

   public UserNotFoundException(String arg0) {
      super(arg0);
   }

   public UserNotFoundException(Throwable arg0) {
      super(arg0);
   }

   public UserNotFoundException(String arg0, Throwable arg1) {
      super(arg0, arg1);
   }

   public UserNotFoundException(String arg0, Throwable arg1, boolean arg2, boolean arg3) {
      super(arg0, arg1, arg2, arg3);
   }

}
```

Robust service interface

```
public interface Directory {
   public User add(User user) throws InvalidUserException, DuplicateUserException, DirectoryException;

   public User findByPhone(long phone) throws UserNotFoundException, DirectoryException;

   public List<User> searchByName(String name) throws DirectoryException;
}
```

Robust service implementation

```java
public class SimpleDirectory implements Directory {
   private Map<Long, User> users = new HashMap<Long, User>();

   public User add(User user) throws InvalidUserException, DuplicateUserException, DirectoryException {
      if (user == null)
         throw new DirectoryException("Invalid User");
      if (user.getName() == null || user.getName().trim().length() < 6)
         throw new InvalidUserException("Invalid name");
      if (user.getPhone() < 0 && Long.toString(user.getPhone()).length() != 10)
         throw new InvalidUserException("Invalid phone number");
      if (user.getEmail() == null || !user.getEmail().contains("@"))
         throw new InvalidUserException("Invalid email");

      if (users.containsKey(user.getPhone()))
         throw new DuplicateUserException("User already exists");

      users.put(user.getPhone(), user);
      return user;
   }

   // TODO: Other methods

}
```

Robust service implementation

```
public class SimpleDirectory implements Directory {
   private Map<Long, User> users = new HashMap<Long, User>();

   // other methods

   public User findByPhone(long phone) throws UserNotFoundException, DirectoryException {
      User result = users.get(phone);
      if (result == null)
         throw new UserNotFoundException();
      return result;
   }

   public List<User> searchByName(String name) throws DirectoryException {
      List<User> results = new ArrayList<User>();
      for (User user : users.values()) {
         if (user.getName().contains(name))
            results.add(user);
      }
      return results;
   }
}
```

Robust Client

```java
public class DirectoryConsole {
   public static void main(String[] args) {
      Directory dir = new SimpleDirectory();
      User user = new User();
      user.setPhone(123L);
      user.setName("Krishna");
      user.setEmail("krishna@glarimy.com");

      try {
         dir.add(user);
         User result = dir.findByPhone(123L);
         System.out.println("Email: " + result.getEmail());
         List<User> results = dir.searchByName("Krishna");
         System.out.println("Number of results: " + results.size());
      } catch (InvalidUserException iue) {
         iue.printStackTrace();
      } catch (DuplicateUserException due) {
         due.printStackTrace();
      } catch (UserNotFoundException unfe) {
         System.out.println("User is not found");
      } catch (DirectoryException de) {
         System.out.println("Internal error. Try again!");
      }
   }
}
```

Glarimy Technology Services
www.glarimy.com

Clean Code Java

Domain Model

Java Beans

```java
public class User {
   private long phone;
   private String name;
   private String email;
   private Address address;

   public long getPhone() {
      return phone;
   }

   public void setPhone(long phone) {
      this.phone = phone;
   }

   public String getName() {
      return name;
   }

   public void setName(String name) {
      this.name = name;
   }

   // other getters and setters

}
```

Java Beans

```java
public class User {
  private long phone;
  private String name;
  private String email;
  private Address address;

  public long getPhone() {
     return phone;
  }

  public void setPhone(long phone) {
     this.phone = phone;
  }

  public String getName() {
     return name;
  }

  public void setName(String name) {
     this.name = name;
  }

  // other getters and setters

}
```

Domain Model: Value Objects

```
public class PhoneNumber {
   private long value;

   public PhoneNumber(long value) throws IllegalValueException {
      if (value < 0 && Long.toString(value).length() != 10)
         throw new IllegalValueException();
      this.value = value;
   }

   public long getValue() {
      return value;
   }
   @Override
   public boolean equals(Object obj) {
      PhoneNumber other = (PhoneNumber) obj;
      return this.value == other.value;
   }

}
```

Domain Model: Value Objects

```
public class PhoneNumber {
   private long value;

   public PhoneNumber(long value) throws IllegalValueException {
      if (value < 0 && Long.toString(value).length() != 10)
         throw new IllegalValueException();
      this.value = value;
   }

   public long getValue() {
      return value;
   }
   @Override
   public boolean equals(Object obj) {
      PhoneNumber other = (PhoneNumber) obj;
      return this.value == other.value;
   }

}
```

Domain Model: Value Objects

```
public class PhoneNumber {
   private long value;

   public PhoneNumber(long value) throws IllegalValueException {
      if (value < 0 && Long.toString(value).length() != 10)
         throw new IllegalValueException();
      this.value = value;
   }

   public long getValue() {
      return value;
   }
   @Override
   public boolean equals(Object obj) {
      PhoneNumber other = (PhoneNumber) obj;
      return this.value == other.value;
   }

}
```

Domain Model: Value Objects

```
public class PhoneNumber {
   private long value;

   public PhoneNumber(long value) throws IllegalValueException {
      if (value < 0 && Long.toString(value).length() != 10)
         throw new IllegalValueException();
      this.value = value;
   }

   public long getValue() {
      return value;
   }
   @Override
   public boolean equals(Object obj) {
      PhoneNumber other = (PhoneNumber) obj;
      return this.value == other.value;
   }

}
```

Domain Model: Value Objects

```
public class PhoneNumber {
   private long value;

   public PhoneNumber(long value) throws IllegalValueException {
      if (value < 0 && Long.toString(value).length() != 10)
         throw new IllegalValueException();
      this.value = value;
   }

   public long getValue() {
      return value;
   }
   @Override
   public boolean equals(Object obj) {
      PhoneNumber other = (PhoneNumber) obj;
      return this.value == other.value;
   }

}
```

Domain Model: Value Objects

```java
public class Name {
   private String value;

   public Name(String value) throws IllegalValueException {
      if (value == null || value.trim().length() < 6)
         throw new IllegalValueException();
      this.value = value;
   }

   public String getValue() {
      return value;
   }

   @Override
   public boolean equals(Object obj) {
      Name other = (Name) obj;
      return this.value.equals(other.value);
   }

}
```

Domain Model: Value Objects

```java
public class Email {
   private String value;

   public Email(String value) throws IllegalValueException {
      if (value == null || !value.contains("@"))
         throw new IllegalValueException();
      this.value = value;
   }

   public String getValue() {
      return value;
   }

   @Override
   public boolean equals(Object obj) {
      Email other = (Email) obj;
      return this.value.equals(other.value);
   }

}
```

Domain Model: Value Objects

```java
public class Address {
   private String location;
   private String city;
   private int pincode;

   public Address(String location, String city, int pincode) {
      // TODO: Validations
      this.location = location;
      this.city = city;
      this.pincode = pincode;
   }

   // getters

   @Override
   public boolean equals(Object obj) {
      Address other = (Address) obj;
      return location.equals(other.location) && city.equals(other.city) && pincode == other.pincode;
   }

}
```

Exception

```java
@SuppressWarnings("serial")
public class IllegalValueException extends Exception {

   public IllegalValueException() {
      super();
   }

   public IllegalValueException(String message, Throwable cause, boolean enableSuppression,
         boolean writableStackTrace) {
      super(message, cause, enableSuppression, writableStackTrace);
   }

   public IllegalValueException(String message, Throwable cause) {
      super(message, cause);
   }

   public IllegalValueException(String message) {
      super(message);
   }

   public IllegalValueException(Throwable cause) {
      super(cause);
   }
}
```

Domain Model: Entities

```
public class User {
   private PhoneNumber phoneNumber;
   private Name name;
   private Email email;
   private Address address;

   public User(PhoneNumber phoneNumber, Name name, Email email) {
      this.phoneNumber = phoneNumber;
      this.name = name;
      this.email = email;
   }

   public PhoneNumber getPhoneNumber() {
      return phoneNumber;
   }

   // TODO: Other getters and setters

}
```

Domain Model: Entities

```
public class User {
  private PhoneNumber phoneNumber;
  private Name name;
  private Email email;
  private Address address;

  public User(PhoneNumber phoneNumber, Name name, Email email) {
    this.phoneNumber = phoneNumber;
    this.name = name;
    this.email = email;
  }

  public PhoneNumber getPhoneNumber() {
    return phoneNumber;
  }

  // TODO: Other getters and setters

}
```

Domain Model: Entities

```
public class User {
   private PhoneNumber phoneNumber;
   private Name name;
   private Email email;
   private Address address;

   public User(PhoneNumber phoneNumber, Name name, Email email) {
      this.phoneNumber = phoneNumber;
      this.name = name;
      this.email = email;
   }

   public PhoneNumber getPhoneNumber() {
      return phoneNumber;
   }

   // TODO: Other getters and setters

}
```

Domain Model: Entities

```java
public class User {
   private PhoneNumber phoneNumber;
   private Name name;
   private Email email;
   private Address address;

   public User(PhoneNumber phoneNumber, Name name, Email email) {
      this.phoneNumber = phoneNumber;
      this.name = name;
      this.email = email;
   }

   public PhoneNumber getPhoneNumber() {
      return phoneNumber;
   }

   // TODO: Other getters and setters

}
```

Domain Model: Service Interface

```
public interface Directory {
   public User add(User user) throws InvalidUserException, DuplicateUserException, DirectoryException;

   public User findByPhone(PhoneNumber phone) throws UserNotFoundException, DirectoryException;

   public List<User> searchByName(Name name) throws DirectoryException;
}
```

Domain Model: Service Interface

```
public interface Directory {
   public User add(User user) throws InvalidUserException, DuplicateUserException, DirectoryException;

   public User findByPhone(PhoneNumber phone) throws UserNotFoundException, DirectoryException;

   public List<User> searchByName(Name name) throws DirectoryException;
}
```

Domain Model: Service Interface

```
public interface Directory {
   public User add(User user) throws InvalidUserException, DuplicateUserException, DirectoryException;

   public User findByPhone(PhoneNumber phone) throws UserNotFoundException, DirectoryException;

   public List<User> searchByName(Name name) throws DirectoryException;
}
```

Domain Model: Service Implementation

```
public class SimpleDirectory implements Directory {
   private Map<PhoneNumber, User> users = new HashMap<PhoneNumber, User>();

   public User add(User user) throws InvalidUserException, DuplicateUserException, DirectoryException {
      if (user == null)
         throw new DirectoryException("Invalid User");
      users.put(user.getPhoneNumber(), user);
      return user;
   }

   public User findByPhone(PhoneNumber phoneNumber) throws UserNotFoundException, DirectoryException {
      User result = users.get(phoneNumber);
      if (result == null)
         throw new UserNotFoundException();
      return result;
   }

   public List<User> searchByName(Name name) throws DirectoryException {
      List<User> results = new ArrayList<User>();
      for (User user : users.values()) {
         if (user.getName().getValue().contains(name.getValue()))
            results.add(user);
      }
      return results;
   }
}
```

Domain Model: Service Implementation

```
public class SimpleDirectory implements Directory {
   private Map<PhoneNumber, User> users = new HashMap<PhoneNumber, User>();

   public User add(User user) throws InvalidUserException, DuplicateUserException, DirectoryException {
      if (user == null)
         throw new DirectoryException("Invalid User");
      users.put(user.getPhoneNumber(), user);
      return user;
   }

   public User findByPhone(PhoneNumber phoneNumber) throws UserNotFoundException, DirectoryException {
      User result = users.get(phoneNumber);
      if (result == null)
         throw new UserNotFoundException();
      return result;
   }

   public List<User> searchByName(Name name) throws DirectoryException {
      List<User> results = new ArrayList<User>();
      for (User user : users.values()) {
         if (user.getName().getValue().contains(name.getValue()))
            results.add(user);
      }
      return results;
   }
}
```

Domain Model: Service Implementation

```
public class SimpleDirectory implements Directory {
   private Map<PhoneNumber, User> users = new HashMap<PhoneNumber, User>();

   public User add(User user) throws InvalidUserException, DuplicateUserException, DirectoryException {
      if (user == null)
         throw new DirectoryException("Invalid User");
      users.put(user.getPhoneNumber(), user);
      return user;
   }

   public User findByPhone(PhoneNumber phoneNumber) throws UserNotFoundException, DirectoryException {
      User result = users.get(phoneNumber);
      if (result == null)
         throw new UserNotFoundException();
      return result;
   }

   public List<User> searchByName(Name name) throws DirectoryException {
      List<User> results = new ArrayList<User>();
      for (User user : users.values()) {
         if (user.getName().getValue().contains(name.getValue()))
            results.add(user);
      }
      return results;
   }
}
```

Domain Model: Service Implementation

```
public class SimpleDirectory implements Directory {
   private Map<PhoneNumber, User> users = new HashMap<PhoneNumber, User>();

   public User add(User user) throws InvalidUserException, DuplicateUserException, DirectoryException {
      if (user == null)
         throw new DirectoryException("Invalid User");
      users.put(user.getPhoneNumber(), user);
      return user;
   }

   public User findByPhone(PhoneNumber phoneNumber) throws UserNotFoundException, DirectoryException {
      User result = users.get(phoneNumber);
      if (result == null)
         throw new UserNotFoundException();
      return result;
   }

   public List<User> searchByName(Name name) throws DirectoryException {
      List<User> results = new ArrayList<User>();
      for (User user : users.values()) {
         if (user.getName().getValue().contains(name.getValue()))
            results.add(user);
      }
      return results;
   }
}
```

Client

```
public class DirectoryConsole {
   public static void main(String[] args) throws Exception {
      Directory dir = new SimpleDirectory();
      PhoneNumber phone = new PhoneNumber(9731423166L);
      Name name = new Name("Krishna");
      Email email = new Email("krishna@glarimy.com");
      User user = new User(phone, name, email);

      dir.add(user);
      User result = dir.findByPhone(phone);
      System.out.println("Email: " + result.getEmail());

      List<User> results = dir.searchByName(name);
      System.out.println("Number of results: " + results.size());
   }
}
```

Client

```java
public class DirectoryConsole {
   public static void main(String[] args) throws Exception {
      Directory dir = new SimpleDirectory();
      PhoneNumber phone = new PhoneNumber(9731423166L);
      Name name = new Name("Krishna");
      Email email = new Email("krishna@glarimy.com");
      User user = new User(phone, name, email);

      dir.add(user);
      User result = dir.findByPhone(phone);
      System.out.println("Email: " + result.getEmail());

      List<User> results = dir.searchByName(name);
      System.out.println("Number of results: " + results.size());
   }
}
```

Client

```
public class DirectoryConsole {
   public static void main(String[] args) throws Exception {
      Directory dir = new SimpleDirectory();
      PhoneNumber phone = new PhoneNumber(9731423166L);
      Name name = new Name("Krishna");
      Email email = new Email("krishna@gtarimy.com");
      User user = new User(phone, name, email);

      dir.add(user);
      User result = dir.findByPhone(phone);
      System.out.println("Email: " + result.getEmail());

      List<User> results = dir.searchByName(name);
      System.out.println("Number of results: " + results.size());
   }
}
```

Client

```
public class DirectoryConsole {
   public static void main(String[] args) throws Exception {
      Directory dir = new SimpleDirectory();
      PhoneNumber phone = new PhoneNumber(9731423166L);
      Name name = new Name("Krishna");
      Email email = new Email("krishna@glarimy.com");
      User user = new User(phone, name, email);

      dir.add(user);
      User result = dir.findByPhone(phone);
      System.out.println("Email: " + result.getEmail());

      List<User> results = dir.searchByName(name);
      System.out.println("Number of results: " + results.size());
   }
}
```

Client

```
public class DirectoryConsole {
   public static void main(String[] args) throws Exception {
      Directory dir = new SimpleDirectory();
      PhoneNumber phone = new PhoneNumber(9731423166L);
      Name name = new Name("Krishna");
      Email email = new Email("krishna@glassimy.com");
      User user = new User(phone, name, email);

      dir.add(user);
      User result = dir.findByPhone(phone);
      System.out.println("Email: " + result.getEmail());

      List<User> results = dir.searchByName(name);
      System.out.println("Number of results: " + results.size());
   }
}
```

Glarimy Technology Services
www.glarimy.com

Clean Code Java

Factory

Client uses interface

```java
public class DirectoryConsole {
   public static void main(String[] args) throws Exception {
      Directory dir = new SimpleDirectory();
      PhoneNumber phone = new PhoneNumber(9731423166L);
      Name name = new Name("Krishna");
      Email email = new Email("krishna@glarimy.com");
      User user = new User(phone, name, email);

      dir.add(user);
      User result = dir.findByPhone(phone);
      System.out.println("Email: " + result.getEmail().getValue());

      List<User> results = dir.searchByName(name);
      System.out.println("Number of results: " + results.size());
   }
}
```

Client uses interface

```
public class DirectoryConsole {
  public static void main(String[] args) throws Exception {
    Directory dir = new SimpleDirectory();
    PhoneNumber phone = new PhoneNumber(9731423166L);
    Name name = new Name("Krishna");
    Email email = new Email("krishna@glarimy.com");
    User user = new User(phone, name, email);

    dir.add(user);
    User result = dir.findByPhone(phone);
    System.out.println("Email: " + result.getEmail().getValue());

    List<User> results = dir.searchByName(name);
    System.out.println("Number of results: " + results.size());
  }
}
```

Factory supplies the object

```
public class ObjectFactory {

   public static SimpleDirectory get() {
      return new SimpleDirectory();
   }

}
```

Factory supplies the object

```
public class ObjectFactory {

   public static SimpleDirectory get() {
      return new SimpleDirectory();
   }

}
```

Factory supplies the object

```
public class ObjectFactory {

   public static SimpleDirectory get() {
      return new SimpleDirectory();
   }

}
```

Client uses the factory

```
public class DirectoryConsole {
   public static void main(String[] args) throws Exception {
      Directory dir = ObjectFactory.get();
      PhoneNumber phone = new PhoneNumber(9731423166L);
      Name name = new Name("Krishna");
      Email email = new Email("krishna@glarimy.com");
      User user = new User(phone, name, email);

      dir.add(user);
      User result = dir.findByPhone(phone);
      System.out.println("Email: " + result.getEmail().getValue());

      List<User> results = dir.searchByName(name);
      System.out.println("Number of results: " + results.size());
   }
}
```

Client uses the factory

```
public class DirectoryConsole {
   public static void main(String[] args) throws Exception {
      Directory dir = ObjectFactory.get();
      PhoneNumber phone = new PhoneNumber(9731423166L);
      Name name = new Name("Krishna");
      Email email = new Email("krishna@glarimy.com");
      User user = new User(phone, name, email);

      dir.add(user);
      User result = dir.findByPhone(phone);
      System.out.println("Email: " + result.getEmail().getValue());

      List<User> results = dir.searchByName(name);
      System.out.println("Number of results: " + results.size());
   }
}
```

Factory supplies objects of an interface

```
public class ObjectFactory {

   public static Directory get() {
      return new SimpleDirectory();
   }

}
```

Factory supplies objects of an interface

```
public class ObjectFactory {

   public static Directory get() {
      return new SimpleDirectory();
   }

}
```

Factory supplies any object

```
public class ObjectFactory {

   public static Object get() {
      return new SimpleDirectory();
   }

}
```

Factory supplies any object

```
public class ObjectFactory {

   public static Object get() {
      return new SimpleDirectory();
   }

}
```

Client uses the factory

```
public class DirectoryConsole {
   public static void main(String[] args) throws Exception {
      Directory dir = (Directory) ObjectFactory.get();
      PhoneNumber phone = new PhoneNumber(9731423166L);
      Name name = new Name("Krishna");
      Email email = new Email("krishna@glarimy.com");
      User user = new User(phone, name, email);

      dir.add(user);
      User result = dir.findByPhone(phone);
      System.out.println("Email: " + result.getEmail().getValue());

      List<User> results = dir.searchByName(name);
      System.out.println("Number of results: " + results.size());
   }
}
```

Client uses the factory

```
public class DirectoryConsole {
   public static void main(String[] args) throws Exception {
      Directory dir = (Directory) ObjectFactory.get();
      PhoneNumber phone = new PhoneNumber(9731423166L);
      Name name = new Name("Krishna");
      Email email = new Email("krishna@glarimy.com");
      User user = new User(phone, name, email);

      dir.add(user);
      User result = dir.findByPhone(phone);
      System.out.println("Email: " + result.getEmail().getValue());

      List<User> results = dir.searchByName(name);
      System.out.println("Number of results: " + results.size());
   }
}
```

Factory applies strategy

```
public class ObjectFactory {

   public static Object get(String key) throws Exception {
      if (key.equals("directory"))
         return new SimpleDirectory();
      throw new Exception("Not supported");
   }

}
```

Factory applies strategy

```
public class ObjectFactory {

   public static Object get(String key) throws Exception {
      if (key.equals("directory"))
         return new SimpleDirectory();
      throw new Exception("Not supported");
   }

}
```

Factory applies strategy

```
public class ObjectFactory {

   public static Object get(String key) throws Exception {
      if (key.equals("directory"))
         return new SimpleDirectory();
      throw new Exception("Not supported");
   }

}
```

Client

```
public class DirectoryConsole {
   public static void main(String[] args) throws Exception {
      Directory dir = (Directory) ObjectFactory.get("directory");
      PhoneNumber phone = new PhoneNumber(9731423166L);
      Name name = new Name("Krishna");
      Email email = new Email("krishna@glarimy.com");
      User user = new User(phone, name, email);

      dir.add(user);
      User result = dir.findByPhone(phone);
      System.out.println("Email: " + result.getEmail().getValue());

      List<User> results = dir.searchByName(name);
      System.out.println("Number of results: " + results.size());
   }
}
```

Client

```
public class DirectoryConsole {
   public static void main(String[] args) throws Exception {
      Directory dir = (Directory) ObjectFactory.get("directory");
      PhoneNumber phone = new PhoneNumber(9731423166L);
      Name name = new Name("Krishna");
      Email email = new Email("krishna@glarimy.com");
      User user = new User(phone, name, email);

      dir.add(user);
      User result = dir.findByPhone(phone);
      System.out.println("Email: " + result.getEmail().getValue());

      List<User> results = dir.searchByName(name);
      System.out.println("Number of results: " + results.size());
   }
}
```

Factory method

```
public interface Factory {

   public Object get(String key) throws Exception;
}
```

Factory method implementation

```
public class ObjectFactory implements Factory {

   @Override
   public Object get(String key) throws Exception {
      if (key.equals("directory"))
         return new SimpleDirectory();
      throw new Exception("Not supported");
   }

}
```

Factory method implementation

```
public class ObjectFactory implements Factory {

   @Override
   public Object get(String key) throws Exception {
      if (key.equals("directory"))
         return new SimpleDirectory();
      throw new Exception("Not supported");
   }

}
```

Client

```java
public class DirectoryConsole {
   public static void main(String[] args) throws Exception {
      Factory factory = new ObjectFactory();

      Directory dir = (Directory) factory.get("directory");
      PhoneNumber phone = new PhoneNumber(9731423166L);
      Name name = new Name("Krishna");
      Email email = new Email("krishna@glarimy.com");
      User user = new User(phone, name, email);

      dir.add(user);
      User result = dir.findByPhone(phone);
      System.out.println("Email: " + result.getEmail().getValue());

      List<User> results = dir.searchByName(name);
      System.out.println("Number of results: " + results.size());
   }
}
```

Client

```
public class DirectoryConsole {
   public static void main(String[] args) throws Exception {
      Factory factory = new ObjectFactory();

      Directory dir = (Directory) factory.get("directory");
      PhoneNumber phone = new PhoneNumber(9731423166L);
      Name name = new Name("Krishna");
      Email email = new Email("krishna@glarimy.com");
      User user = new User(phone, name, email);

      dir.add(user);
      User result = dir.findByPhone(phone);
      System.out.println("Email: " + result.getEmail().getValue());

      List<User> results = dir.searchByName(name);
      System.out.println("Number of results: " + results.size());
   }
}
```

Client

```
public class DirectoryConsole {
   public static void main(String[] args) throws Exception {
      Factory factory = new ObjectFactory();

      Directory dir = (Directory) factory.get("directory");
      PhoneNumber phone = new PhoneNumber(9731423166L);
      Name name = new Name("Krishna");
      Email email = new Email("krishna@glarimy.com");
      User user = new User(phone, name, email);

      dir.add(user);
      User result = dir.findByPhone(phone);
      System.out.println("Email: " + result.getEmail().getValue());

      List<User> results = dir.searchByName(name);
      System.out.println("Number of results: " + results.size());
   }
}
```

Factory uses configuration

```
public class ObjectFactory implements Factory {
   private Properties config;

   public ObjectFactory() {
      config = new Properties();
      config.put("directory", "com.glarimy.SimpleDirectory");
   }

   @Override
   public Object get(String key) throws Exception {
      String className = config.getProperty(key);
      @SuppressWarnings("rawtypes")
      Class claz = Class.forName(className);
      Object o = claz.newInstance();
      return o;
   }

}
```

Factory uses configuration

```
public class ObjectFactory implements Factory {
   private Properties config;

   public ObjectFactory() {
      config = new Properties();
      config.put("directory", "com.glarimy.SimpleDirectory");
   }

   @Override
   public Object get(String key) throws Exception {
      String className = config.getProperty(key);
      @SuppressWarnings("rawtypes")
      Class claz = Class.forName(className);
      Object o = claz.newInstance();
      return o;
   }

}
```

Factory uses configuration

```
public class ObjectFactory implements Factory {
   private Properties config;

   public ObjectFactory() {
      config = new Properties();
      config.put("directory", "com.glarimy.SimpleDirectory");
   }

   @Override
   public Object get(String key) throws Exception {
      String className = config.getProperty(key);
      @SuppressWarnings("rawtypes")
      Class claz = Class.forName(className);
      Object o = claz.newInstance();
      return o;
   }

}
```

Factory uses configuration

```
public class ObjectFactory implements Factory {
   private Properties config;

   public ObjectFactory() {
      config = new Properties();
      config.put("directory", "com.glarimy.SimpleDirectory");
   }

   @Override
   public Object get(String key) throws Exception {
      String className = config.getProperty(key);
      @SuppressWarnings("rawtypes")
      Class claz = Class.forName(className);
      Object o = claz.newInstance();
      return o;
   }

}
```

Factory uses external configuration

```
public class ObjectFactory implements Factory {
   private Properties config;

   public ObjectFactory() throws Exception {
      config = new Properties();
      config.load(new FileReader("config.properties"));
   }

   @Override
   public Object get(String key) throws Exception {
      String className = config.getProperty(key);
      @SuppressWarnings("rawtypes")
      Class claz = Class.forName(className);
      Object o = claz.newInstance();
      return o;
   }

}
```

Factory uses external configuration

```
public class ObjectFactory implements Factory {
   private Properties config;

   public ObjectFactory() throws Exception {
      config = new Properties();
      config.load(new FileReader("config.properties"));
   }

   @Override
   public Object get(String key) throws Exception {
      String className = config.getProperty(key);
      @SuppressWarnings("rawtypes")
      Class claz = Class.forName(className);
      Object o = claz.newInstance();
      return o;
   }

}
```

Configuration

```
directory=com.glarimy.SimpleDirectory
```

Glarimy Technology Services
www.glarimy.com

Clean Code Java

Adapter | Repository

Domain Model: Service Implementation

```
public class SimpleDirectory implements Directory {
   private Map<PhoneNumber, User> users = new HashMap<PhoneNumber, User>();

   public User add(User user) throws InvalidUserException, DuplicateUserException, DirectoryException {
      if (user == null)
         throw new DirectoryException("Invalid User");
      users.put(user.getPhoneNumber(), user);
      return user;
   }

   public User findByPhone(PhoneNumber phoneNumber) throws UserNotFoundException, DirectoryException {
      User result = users.get(phoneNumber);
      if (result == null)
         throw new UserNotFoundException();
      return result;
   }

   public List<User> searchByName(Name name) throws DirectoryException {
      List<User> results = new ArrayList<User>();
      for (User user : users.values()) {
         if (user.getName().getValue().contains(name.getValue()))
            results.add(user);
      }
      return results;
   }
}
```

Domain Model: Service Implementation

```
public class SimpleDirectory implements Directory {
   private Map<PhoneNumber, User> users = new HashMap<PhoneNumber, User>();

   public User add(User user) throws InvalidUserException, DuplicateUserException, DirectoryException {
      if (user == null)
         throw new DirectoryException("Invalid User");
      users.put(user.getPhoneNumber(), user);
      return user;
   }

   public User findByPhone(PhoneNumber phoneNumber) throws UserNotFoundException, DirectoryException {
      User result = users.get(phoneNumber);
      if (result == null)
         throw new UserNotFoundException();
      return result;
   }

   public List<User> searchByName(Name name) throws DirectoryException {
      List<User> results = new ArrayList<User>();
      for (User user : users.values()) {
         if (user.getName().getValue().contains(name.getValue()))
            results.add(user);
      }
      return results;
   }
}
```

Storage adapter

```
public interface Storage {
   public User save(User user);
   public User findById(PhoneNumber id);
   public List<User> findByName(Name name);
}
```

An implementation of storage adapter

```
public class InMemoryStorage implements Storage {
   private Map<PhoneNumber, User> users = new HashMap<PhoneNumber, User>();

   @Override
   public User save(User user) {
      users.put(user.getPhoneNumber(), user);
      return user;
   }

   @Override
   public User findById(PhoneNumber id) {
      return users.get(id);
   }

   @Override
   public List<User> findByName(Name name) {
      List<User> results = new ArrayList<User>();
      for (User user : users.values()) {
         if (user.getName().getValue().contains(name.getValue()))
            results.add(user);
      }
      return results;

   }

}
```

An implementation of storage adapter

```
public class InMemoryStorage implements Storage {
   private Map<PhoneNumber, User> users = new HashMap<PhoneNumber, User>();

   @Override
   public User save(User user) {
      users.put(user.getPhoneNumber(), user);
      return user;
   }

   @Override
   public User findById(PhoneNumber id) {
      return users.get(id);
   }

   @Override
   public List<User> findByName(Name name) {
      List<User> results = new ArrayList<User>();
      for (User user : users.values()) {
         if (user.getName().getValue().contains(name.getValue()))
            results.add(user);
      }
      return results;

   }

}
```

Service uses a storage adapter

```
public class SimpleDirectory implements Directory {
   private Storage store;

   public SimpleDirectory(Storage store) {
      this.store = store;
   }

   public User add(User user) throws InvalidUserException, DuplicateUserException, DirectoryException {
      if (user == null)
         throw new DirectoryException("Invalid User");
      store.save(user);
      return user;
   }

   public User findByPhone(PhoneNumber phoneNumber) throws UserNotFoundException, DirectoryException {
      User result = store.findById(phoneNumber);
      if (result == null)
         throw new UserNotFoundException();
      return result;
   }

   public List<User> searchByName(Name name) throws DirectoryException {
      return store.findByName(name);
   }
}
```

Service uses a storage adapter

```
public class SimpleDirectory implements Directory {
  private Storage store;

  public SimpleDirectory(Storage store) {
    this.store = store;
  }

  public User add(User user) throws InvalidUserException, DuplicateUserException, DirectoryException {
    if (user == null)
      throw new DirectoryException("Invalid User");
    store.save(user);
    return user;
  }

  public User findByPhone(PhoneNumber phoneNumber) throws UserNotFoundException, DirectoryException {
    User result = store.findById(phoneNumber);
    if (result == null)
      throw new UserNotFoundException();
    return result;
  }

  public List<User> searchByName(Name name) throws DirectoryException {
    return store.findByName(name);
  }
}
```

Service uses a storage adapter

```
public class SimpleDirectory implements Directory {
  private Storage store;

  public SimpleDirectory(Storage store) {
    this.store = store;
  }

  public User add(User user) throws InvalidUserException, DuplicateUserException, DirectoryException {
    if (user == null)
      throw new DirectoryException("Invalid User");
    store.save(user);
    return user;
  }

  public User findByPhone(PhoneNumber phoneNumber) throws UserNotFoundException, DirectoryException {
    User result = store.findById(phoneNumber);
    if (result == null)
      throw new UserNotFoundException();
    return result;
  }

  public List<User> searchByName(Name name) throws DirectoryException {
    return store.findByName(name);
  }
}
```

Service uses a storage adapter

```
public class SimpleDirectory implements Directory {
  private Storage store;

  public SimpleDirectory(Storage store) {
    this.store = store;
  }

  public User add(User user) throws InvalidUserException, DuplicateUserException, DirectoryException {
    if (user == null)
      throw new DirectoryException("Invalid User");
    store.save(user);
    return user;
  }

  public User findByPhone(PhoneNumber phoneNumber) throws UserNotFoundException, DirectoryException {
    User result = store.findById(phoneNumber);
    if (result == null)
      throw new UserNotFoundException();
    return result;
  }

  public List<User> searchByName(Name name) throws DirectoryException {
    return store.findByName(name);
  }
}
```

Factory injects the dependencies

```
public class ObjectFactory implements Factory {

   @Override
   public Object get(String key) throws Exception {
      if (key.equals("directory")) {
         Storage repo = (Storage) get("storage");
         Directory dir = new SimpleDirectory(repo);
         return dir;
      }
      if (key.equals("storage")) {
         return new InMemoryStorage();
      }
      throw new Exception("Not supported");
   }

}
```

Factory injects the dependencies

```
public class ObjectFactory implements Factory {

   @Override
   public Object get(String key) throws Exception {
      if (key.equals("directory")) {
         Storage repo = (Storage) get("storage");
         Directory dir = new SimpleDirectory(repo);
         return dir;
      }
      if (key.equals("storage")) {
         return new InMemoryStorage();
      }
      throw new Exception("Not supported");
   }

}
```

Factory injects the dependencies

```
public class ObjectFactory implements Factory {

   @Override
   public Object get(String key) throws Exception {
      if (key.equals("directory")) {
         Storage storage = (Storage) get("storage");
         Directory dir = new SimpleDirectory(storage);
         return dir;
      }
      if (key.equals("storage")) {
         return new InMemoryStorage();
      }
      throw new Exception("Not supported");
   }

}
```

Client

```
public class DirectoryConsole {
   public static void main(String[] args) throws Exception {
      Factory factory = new ObjectFactory();
      Directory dir = (Directory) factory.get("directory");
      PhoneNumber phone = new PhoneNumber(9731423166L);
      Name name = new Name("Krishna");
      Email email = new Email("krishna@glarimy.com");
      User user = new User(phone, name, email);

      dir.add(user);
      User result = dir.findByPhone(phone);
      System.out.println(">>>> Found user with email: " + result.getEmail().getValue());

      List<User> results = dir.searchByName(name);
      System.out.println(">>>> Found " + results.size() + " result(s)");
   }
}
```

Glarimy Technology Services
www.glarimy.com

Clean Code Java

Proxy | Decorator | Chain of Responsibility

A proxy to log invocations on the directory

```
public class Logger implements Directory {
   private Directory target;

   public Logger(Directory target) {
      this.target = target;
   }

   @Override
   public User add(User user) throws InvalidUserException, DuplicateUserException, DirectoryException {
      System.out.println(new Date() + " - adding user: " + user.getPhoneNumber().getValue());
      User result = target.add(user);
      System.out.println(new Date() + " - added user: " + user.getPhoneNumber().getValue());
      return result;
   }

   // other methods

}
```

A proxy to log invocations on the directory

```java
public class Logger implements Directory {
   private Directory target;

   public Logger(Directory target) {
      this.target = target;
   }

   @Override
   public User add(User user) throws InvalidUserException, DuplicateUserException, DirectoryException {
      System.out.println(new Date() + " - adding user: " + user.getPhoneNumber().getValue());
      User result = target.add(user);
      System.out.println(new Date() + " - added user: " + user.getPhoneNumber().getValue());
      return result;
   }

   // other methods

}
```

A proxy to log invocations on the directory

```java
public class Logger implements Directory {
  private Directory target;

  public Logger(Directory target) {
    this.target = target;
  }

  @Override
  public User add(User user) throws InvalidUserException, DuplicateUserException, DirectoryException {
    System.out.println(new Date() + " - adding user: " + user.getPhoneNumber().getValue());
    User result = target.add(user);
    System.out.println(new Date() + " - added user: " + user.getPhoneNumber().getValue());
    return result;
  }

  // other methods

}
```

A proxy to log invocations on the directory

```
public class Logger implements Directory {
   private Directory target;

   public Logger(Directory target) {
      this.target = target;
   }

   @Override
   public User add(User user) throws InvalidUserException, DuplicateUserException, DirectoryException {
      System.out.println(new Date() + " - adding user: " + user.getPhoneNumber().getValue());
      User result = target.add(user);
      System.out.println(new Date() + " - added user: " + user.getPhoneNumber().getValue());
      return result;
   }

   // other methods

}
```

A proxy to log invocations on the directory

```
public class Logger implements Directory {
   private Directory target;

   public Logger(Directory target) {
      this.target = target;
   }

   @Override
   public User add(User user) throws InvalidUserException, DuplicateUserException, DirectoryException {
      System.out.println(new Date() + " - adding user: " + user.getPhoneNumber().getValue());
      User result = target.add(user);
      System.out.println(new Date() + " - added user: " + user.getPhoneNumber().getValue());
      return result;
   }

   // other methods

}
```

A proxy to log invocations on the directory

```
public class Logger implements Directory {
   private Directory target;

   public Logger(Directory target) {
      this.target = target;
   }

   @Override
   public User add(User user) throws InvalidUserException, DuplicateUserException, DirectoryException {
      System.out.println(new Date() + " - adding user: " + user.getPhoneNumber().getValue());
      User result = target.add(user);
      System.out.println(new Date() + " - added user: " + user.getPhoneNumber().getValue());
      return result;
   }

   // other methods

}
```

A proxy to log invocations on the directory

```
public class Logger implements Directory {
   private Directory target;

   public Logger(Directory target) {
      this.target = target;
   }

   @Override
   public User add(User user) throws InvalidUserException, DuplicateUserException, DirectoryException {
      System.out.println(new Date() + " - adding user: " + user.getPhoneNumber().getValue());
      User result = target.add(user);
      System.out.println(new Date() + " - added user: " + user.getPhoneNumber().getValue());
      return result;
   }

   // other methods

}
```

A proxy to log invocations on the directory

```
public class Logger implements Directory {
   private Directory target;

   public Logger(Directory target) {
      this.target = target;
   }

   // other methods

   @Override
   public User findByPhone(PhoneNumber phoneNumber) throws UserNotFoundException, DirectoryException {
      System.out.println(new Date() + " - finding user: " + phoneNumber.getValue());
      User user = target.findByPhone(phoneNumber);
      System.out.println(new Date() + " - found user: " + phoneNumber.getValue());
      return user;
   }

   @Override
   public List<User> searchByName(Name name) throws DirectoryException {
      System.out.println(new Date() + " - searching user: " + name.getValue());
      List<User> users = target.searchByName(name);
      System.out.println(new Date() + " - searched user: " + name.getValue());
      return users;
   }

}
```

A proxy to log invocations on the directory

```
public class Logger implements Directory {
   private Directory target;

   public Logger(Directory target) {
      this.target = target;
   }

   // other methods

   @Override
   public User findByPhone(PhoneNumber phoneNumber) throws UserNotFoundException, DirectoryException {
      System.out.println(new Date() + " - finding user: " + phoneNumber.getValue());
      User user = target.findByPhone(phoneNumber);
      System.out.println(new Date() + " - found user: " + phoneNumber.getValue());
      return user;
   }

   @Override
   public List<User> searchByName(Name name) throws DirectoryException {
      System.out.println(new Date() + " - searching user: " + name.getValue());
      List<User> users = target.searchByName(name);
      System.out.println(new Date() + " - searched user: " + name.getValue());
      return users;
   }

}
```

Factory decorates objects with proxy

```
public class ObjectFactory implements Factory {
   private boolean logging;

   public ObjectFactory() throws Exception {
      Properties config = new Properties();
      config.load(new FileReader("config.properties"));
      this.logging = Boolean.parseBoolean(config.getProperty("logging.enabled"));
   }

   @Override
   public Object get(String key) throws Exception {
      if (key.equals("directory")) {
         Storage repo = (Storage) get("storage");
         Directory dir = new SimpleDirectory(repo);
         if (logging)
            dir = new Logger(dir);
         return dir;
      }
      if (key.equals("storage"))
         return new InMemoryStorage();

      throw new Exception("Not supported");
   }

}
```

Factory decorates objects with proxy

```
public class ObjectFactory implements Factory {
   private boolean logging;

   public ObjectFactory() throws Exception {
      Properties config = new Properties();
      config.load(new FileReader("config.properties"));
      this.logging = Boolean.parseBoolean(config.getProperty("logging.enabled"));
   }

   @Override
   public Object get(String key) throws Exception {
      if (key.equals("directory")) {
         Storage repo = (Storage) get("storage");
         Directory dir = new SimpleDirectory(repo);
         if (logging)
            dir = new Logger(dir);
         return dir;
      }
      if (key.equals("storage"))
         return new InMemoryStorage();

      throw new Exception("Not supported");
   }

}
```

Configuration

```
logging.enabled=true
```

A proxy for collecting performance metrics

```
public class MetricsCollector implements Directory {
   private Directory target;

   public MetricsCollector(Directory target) {
      this.target = target;
   }

   @Override
   public User add(User user) throws InvalidUserException, DuplicateUserException, DirectoryException {
      long start = new Date().getTime();
      User result = target.add(user);
      System.out.println("Metrics: added user in " + (new Date().getTime() - start) + " ms");
      return result;
   }

   // other methods

}
```

A proxy for collecting performance metrics

```
public class MetricsCollector implements Directory {
   private Directory target;

   public MetricsCollector(Directory target) {
      this.target = target;
   }

   // other methods

   @Override
   public User findByPhone(PhoneNumber phoneNumber) throws UserNotFoundException, DirectoryException {
      long start = new Date().getTime();
      User user = target.findByPhone(phoneNumber);
      System.out.println("Metrics: found user in " + (new Date().getTime() - start) + " ms");
      return user;
   }

   @Override
   public List<User> searchByName(Name name) throws DirectoryException {
      long start = new Date().getTime();
      List<User> users = target.searchByName(name);
      System.out.println("Metrics: found user in " + (new Date().getTime() - start) + " ms");
      return users;
   }

}
```

Factory builds a chain of responsibility

```
public class ObjectFactory implements Factory {
   private boolean logging;
   private boolean metrics;

   public ObjectFactory() throws Exception {
      Properties config = new Properties();
      config.load(new FileReader("config.properties"));
      this.logging = Boolean.parseBoolean(config.getProperty("logging.enabled"));
      this.metrics = Boolean.parseBoolean(config.getProperty("metrics.enabled"));
   }

   @Override
   public Object get(String key) throws Exception {
      if (key.equals("directory")) {
         Storage repo = (Storage) get("storage");
         Directory dir = new SimpleDirectory(repo);
         if (logging)
            dir = new Logger(dir);
         if (metrics)
            dir = new MetricsCollector(dir);
         return dir;
      }
      if (key.equals("storage"))
         return new InMemoryStorage();
      throw new Exception("Not supported");
   }

}
```

Factory builds a chain of responsibility

```
public class ObjectFactory implements Factory {
  private boolean logging;
  private boolean metrics;

  public ObjectFactory() throws Exception {
    Properties config = new Properties();
    config.load(new FileReader("config.properties"));
    this.logging = Boolean.parseBoolean(config.getProperty("logging.enabled"));
    this.metrics = Boolean.parseBoolean(config.getProperty("metrics.enabled"));
  }

  @Override
  public Object get(String key) throws Exception {
    if (key.equals("directory")) {
      Storage repo = (Storage) get("storage");
      Directory dir = new SimpleDirectory(repo);
      if (logging)
        dir = new Logger(dir);
      if (metrics)
        dir = new MetricsCollector(dir);
      return dir;
    }
    if (key.equals("storage"))
      return new InMemoryStorage();
    throw new Exception("Not supported");
  }

}
```

Glarimy Technology Services
www.glarimy.com

Clean Code Java

Application Layer

Data Transfer Objects

```
public class UserData {
   public String name;
   public long phoneNumber;
   public String email;
   public String location;
   public String city;
   public int pincode;

   public UserData() {

   }

   public UserData(User user) {
      name = user.getName().getValue();
      phoneNumber = user.getPhoneNumber().getValue();
      email = user.getEmail().getValue();
      location = user.getAddress().getLocation();
      city = user.getAddress().getCity();
      pincode = user.getAddress().getPincode();
   }

   public User toUser() throws IllegalValueException {
      User user = new User(new PhoneNumber(phoneNumber), new Name(name), new Email(email));
      user.setAddress(new Address(location, city, pincode));
      return user;

   }
}
```

Service Entry Point

```
public class DirectoryController {

   private Directory directory;

   public DirectoryController(Directory directory) {
      this.directory = directory;
   }

   public UserData register(UserData data) throws Exception {
      User user = data.toUser();
      User registeredUser = directory.add(user);
      return new UserData(registeredUser);
   }

   public UserData findByPhone(long phone) throws Exception {
      User user = directory.findByPhone(new PhoneNumber(phone));
      return new UserData(user);

   }

   public List<UserData> findByName(String name) throws Exception {
      List<User> users = directory.searchByName(new Name(name));
      List<UserData> list = new ArrayList<UserData>();
      for (User user : users)
         list.add(new UserData(user));
      return list;
   }

}
```

Factory

```java
public class ObjectFactory implements Factory {
   private boolean logging;
   private boolean metrics;

   public ObjectFactory() throws Exception {
      Properties config = new Properties();
      config.load(new FileReader("config.properties"));
      this.logging = Boolean.parseBoolean(config.getProperty("logging.enabled"));
      this.metrics = Boolean.parseBoolean(config.getProperty("metrics.enabled"));
   }

   @Override
   public Object get(String key) throws Exception {
      if (key.equals("controller")) {
         Storage repo = (Storage) get("storage");
         Directory dir = new SimpleDirectory(repo);
         if (logging)
            dir = new Logger(dir);
         if (metrics)
            dir = new MetricsCollector(dir);
         return new DirectoryController(dir);
      }
      if (key.equals("storage")) {
         return new InMemoryStorage();
      }
      throw new Exception("Not supported");
   }

}
```

Factory

```java
public class DirectoryConsole {
   public static void main(String[] args) throws Exception {
      Factory factory = new ObjectFactory();
      DirectoryController dir = (DirectoryController) factory.get("controller");
      UserData user = new UserData();
      user.name = "Krishna";
      user.phoneNumber = 9731423166L;
      user.email = "krishna@glarimy.com";
      user.location = "Pai Layaout";
      user.city = "Bengaluru";
      user.pincode = 534101;

      dir.register(user);
      UserData result = dir.findByPhone(9731423166L);
      System.out.println(">>>> Found user with email: " + result.email);

      List<UserData> results = dir.findByName("Krishna");
      System.out.println(">>>> Found " + results.size() + " result(s)");
   }
}
```

Glarimy Technology Services
www.glarimy.com

Clean Code Java

Functional Approach

Optional Returns and Functional Interfaces

```
public interface Storage {
   public User save(User user);
   public Optional<User> findById(PhoneNumber id);
   public List<User> search(Predicate<User> condition);
}
```

Stream Processing

```
public class InMemoryStorage implements Storage {
   private Map<PhoneNumber, User> users = new HashMap<PhoneNumber, User>();

   @Override
   public User save(User user) {
      users.put(user.getPhoneNumber(), user);
      return user;
   }

   @Override
   public Optional<User> findById(PhoneNumber id) {
      User user = users.get(id);
      if (user == null)
         return Optional.empty();
      return Optional.of(user);
   }

   @Override
   public List<User> search(Predicate<User> condition) {
      return users.values().stream().filter(condition).collect(Collectors.toList());
   }

}
```

Functional Interfaces

```
public interface Directory {
   public User add(User user) throws InvalidUserException, DuplicateUserException, DirectoryException;

   public Optional<User> findByPhone(PhoneNumber phone) throws DirectoryException;

   public List<User> searchByCondition(Predicate<User> condition) throws DirectoryException;
}
```

Implementation

```java
public class SimpleDirectory implements Directory {
   private Storage store;

   public SimpleDirectory(Storage store) {
      this.store = store;
   }

   public User add(User user) throws InvalidUserException, DuplicateUserException, DirectoryException {
      if (user == null)
         throw new DirectoryException("Invalid User");
      store.save(user);
      return user;
   }

   public Optional<User> findByPhone(PhoneNumber phoneNumber) throws DirectoryException {
      return store.findById(phoneNumber);
   }

   public List<User> searchByCondition(Predicate<User> predicate) throws DirectoryException {
      return store.search(predicate);
   }
}
```

Refactored Proxies

```
public class Logger implements Directory {
   private Directory target;

   public Logger(Directory target) {
      this.target = target;
   }

   @Override
   public User add(User user) throws InvalidUserException, DuplicateUserException, DirectoryException {
      System.out.println(new Date() + " - adding user: " + user.getPhoneNumber().getValue());
      User result = target.add(user);
      System.out.println(new Date() + " - added user: " + user.getPhoneNumber().getValue());
      return result;
   }

   @Override
   public Optional<User> findByPhone(PhoneNumber phoneNumber) throws DirectoryException {
      System.out.println(new Date() + " - finding user: " + phoneNumber.getValue());
      Optional<User> user = target.findByPhone(phoneNumber);
      System.out.println(new Date() + " - found user: " + phoneNumber.getValue());
      return user;
   }

   @Override
   public List<User> searchByCondition(Predicate<User> condition) throws DirectoryException {
      System.out.println(new Date() + " - searching users ");
      List<User> users = target.searchByCondition(condition);
      System.out.println(new Date() + " - searched user ");
      return users;
   }

}
```

Refactored Proxies

```
public class MetricsCollector implements Directory {
   private Directory target;

   public MetricsCollector(Directory target) {
      this.target = target;
   }

   @Override
   public User add(User user) throws InvalidUserException, DuplicateUserException, DirectoryException {
      long start = new Date().getTime();
      User result = target.add(user);
      System.out.println("Metrics: added user in " + (new Date().getTime() - start) + " ms");
      return result;
   }

   @Override
   public Optional<User> findByPhone(PhoneNumber phoneNumber) throws UserNotFoundException, DirectoryException {
      long start = new Date().getTime();
      Optional<User> user = target.findByPhone(phoneNumber);
      System.out.println("Metrics: found user in " + (new Date().getTime() - start) + " ms");
      return user;
   }

   @Override
   public List<User> searchByCondition(Predicate<User> condition) throws DirectoryException {
      long start = new Date().getTime();
      List<User> users = target.searchByCondition(condition);
      System.out.println("Metrics: found user in " + (new Date().getTime() - start) + " ms");
      return users;
   }

}
```

Controller

```java
public class DirectoryController {

   private Directory directory;

   public DirectoryController(Directory directory) {
      this.directory = directory;
   }

   public UserData register(UserData data) throws Exception {
      User user = data.toUser();
      User registeredUser = directory.add(user);
      return new UserData(registeredUser);
   }

   public Optional<UserData> findByPhone(long phone) throws Exception {
      try {
         User user = directory.findByPhone(new PhoneNumber(phone)).orElseThrow(() -> new UserNotFoundException());
         return Optional.of(new UserData(user));
      } catch (UserNotFoundException e) {
         return Optional.empty();
      }
   }

   public List<UserData> findByName(String name) throws Exception {
      List<User> users = directory.searchByCondition(u -> u.getName().getValue().contains(name));
      List<UserData> list = new ArrayList<UserData>();
      for (User user : users)
         list.add(new UserData(user));
      return list;
   }

}
```

Client

```java
public class DirectoryConsole {
   public static void main(String[] args) throws Exception {
      Factory factory = new ObjectFactory();
      DirectoryController dir = (DirectoryController) factory.get("controller");
      UserData user = new UserData();
      user.name = "Krishna";
      user.phoneNumber = 9731423166L;
      user.email = "krishna@glarimy.com";
      user.location = "Pai Layaout";
      user.city = "Bengaluru";
      user.pincode = 534101;

      dir.register(user);
      dir.findByPhone(9731423166L).ifPresent(u -> System.out.println(">>>> Found user with email: " + u.email));

      List<UserData> results = dir.findByName("Krishna");
      System.out.println(">>>> Found " + results.size() + " result(s)");
   }
}
```

Glarimy Technology Services
www.glarimy.com

Clean Code Java

A Spring Illustration

A Spring Illustration

```
@SuppressWarnings("serial")
@Embeddable
public class PhoneNumber implements Serializable {
   @Column(name = "phone_number")
   private long value;

   public PhoneNumber() {

   }

   public PhoneNumber(long value) throws IllegalValueException {
      if (value < 0 && Long.toString(value).length() != 10)
         throw new IllegalValueException();
      this.value = value;
   }

   public long getValue() {
      return value;
   }

   @Override
   public boolean equals(Object obj) {
      PhoneNumber other = (PhoneNumber) obj;
      return this.value == other.value;
   }

   @Override
   public String toString() {
      return "PhoneNumber [value=" + value + "]";
   }

}
```

A Spring Illustration

```java
@Embeddable
public class Name {
   @Column(name = "name")
   private String value;

   Name() {

   }

   public Name(String value) throws IllegalValueException {
      if (value == null || value.trim().length() < 6)
         throw new IllegalValueException();
      this.value = value;
   }

   public String getValue() {
      return value;
   }

   @Override
   public boolean equals(Object obj) {
      Name other = (Name) obj;
      return this.value.equals(other.value);
   }

   @Override
   public String toString() {
      return "Name [value=" + value + "]";
   }

}
```

A Spring Illustration

```java
@Embeddable
public class Email {
   @Column(name = "email")
   private String value;

   Email() {

   }

   public Email(String value) throws IllegalValueException {
      if (value == null || !value.contains("@"))
         throw new IllegalValueException();
      this.value = value;
   }

   public String getValue() {
      return value;
   }

   @Override
   public boolean equals(Object obj) {
      Email other = (Email) obj;
      return this.value.equals(other.value);
   }

   @Override
   public String toString() {
      return "Email [value=" + value + "]";
   }

}
```

A Spring Illustration

```java
@Embeddable
public class Address {
   private String location;
   private String city;
   private int pincode;

   Address() {

   }

   public Address(String location, String city, int pincode) {
      super();
      this.location = location;
      this.city = city;
      this.pincode = pincode;
   }

   // getters

   @Override
   public boolean equals(Object obj) {
      Address other = (Address) obj;
      return location.equals(other.location) && city.equals(other.city) && pincode == other.pincode;
   }

   @Override
   public String toString() {
      return "Address [location=" + location + ", city=" + city + ", pincode=" + pincode + "]";
   }

}
```

A Spring Illustration

```
@Entity
public class User {
   @EmbeddedId
   private PhoneNumber phoneNumber;
   private Name name;
   private Email email;
   private Address address;

   User() {

   }

   public User(PhoneNumber phoneNumber, Name name, Email email) {
      super();
      this.phoneNumber = phoneNumber;
      this.name = name;
      this.email = email;
   }

   public PhoneNumber getPhoneNumber() {
      return phoneNumber;
   }

   // other getters and setters

   @Override
   public String toString() {
      return "User [phoneNumber=" + phoneNumber + ", name=" + name + ", email=" + email + ", address=" + address
            + "]";
   }
}
```

A Spring Illustration

```
@Repository
public interface Storage extends JpaRepository<User, PhoneNumber> {
   public List<User> findByName(Name name);
}
```

A Spring Illustration

```java
@Service
public class SimpleDirectory implements Directory {
   @Autowired
   private Storage store;

   public User add(User user) throws InvalidUserException, DuplicateUserException, DirectoryException {
      if (user == null)
         throw new DirectoryException("Invalid User");
      store.save(user);
      return user;
   }

   public Optional<User> findByPhone(PhoneNumber phoneNumber) throws DirectoryException {
      return store.findById(phoneNumber);
   }

   public List<User> searchByName(Name name) throws DirectoryException {
      return store.findByName(name);
   }
}
```

A Spring Illustration

```java
@Aspect
@Component
@Order(1)
public class MetricsCollector {

   @Around("execution (* com.glarimy.domain.SimpleDirectory.*(..))")
   public Object measure(ProceedingJoinPoint p) throws Throwable {
      long start = new Date().getTime();
      Object result = p.proceed();
      System.out.println(p.getSignature() + " is completed in " + (new Date().getTime() - start) + " ms");
      return result;
   }

}
```

A Spring Illustration

```
@Aspect
@Component
@Order(2)
public class Logger {

   @Around("execution (* com.glarimy.domain.SimpleDirectory.*(..))")
   public Object log(ProceedingJoinPoint p) throws Throwable {
      System.out.println("Entering " + p.getSignature() + " with " + p.getArgs()[0]);
      Object result = p.proceed();
      System.out.println("Leaving  " + p.getSignature() + " with " + result);
      return result;
   }

}
```

A Spring Illustration

```java
@RestController
public class DirectoryController {

   @Autowired
   private Directory directory;

   @PostMapping("/user")
   public ResponseEntity<UserData> register(@RequestBody UserData data) throws Exception {
      User user = data.toUser();
      User registeredUser = directory.add(user);
      return new ResponseEntity<UserData>(new UserData(registeredUser), HttpStatus.CREATED);
   }

   @GetMapping("/user/{phone}")
   public ResponseEntity<UserData> findByPhone(@PathVariable("phone") long phone) throws Exception {
      try {
         User user = directory.findByPhone(new PhoneNumber(phone)).orElseThrow(() -> new UserNotFoundException());
         return new ResponseEntity<UserData>(new UserData(user), HttpStatus.OK);
      } catch (UserNotFoundException e) {
         return new ResponseEntity<UserData>(HttpStatus.NOT_FOUND);
      }
   }

   @GetMapping("/user")
   public ResponseEntity<List<UserData>> findByName(@RequestParam("name") String name) throws Exception {
      List<User> users = directory.searchByName(new Name(name));
      List<UserData> list = new ArrayList<UserData>();
      for (User user : users)
         list.add(new UserData(user));
      return new ResponseEntity<List<UserData>>(list, HttpStatus.OK);
   }

}
```

A Spring Illustration

```
server.port=8080
spring.application.name=bank
spring.datasource.url=jdbc:mysql://localhost:3306/dir?useSSL=false
spring.datasource.username=root
spring.datasource.password=adminadmin
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQL5Dialect
spring.jpa.hibernate.ddl-auto=update
```

A Spring Illustration

```
@SpringBootApplication
public class DirectoryBootstrapper {
   public static void main(String[] args) {
      SpringApplication.run(DirectoryBootstrapper.class, args);
   }
}
```