Lecture 2

Introduction to Spring Boot

Software Reuse - Application frameworks

Topics covered

- ✧ Introduction to Spring Boot
  - Maven
  - Thymeleaf
  - Spring Data JPA
  - Basic Spring MVC
- ✧ Software Reuse
  - Application Frameworks

Introduction to Maven

What is Maven?

✧ A build tool and dependencies management tool

- Comes built-in in many IDEs, including IntelliJ IDEA
- Is one of the two choices for dependencies management in a Spring application (the other one is Gradle)

✧ Maven project structure:

- `pom.xml` : application configuration file
- `src/main/java` : Application sources
- `src/main/resources` : Application resources
- `src/test/java` : Test sources
- `src/test/resources` : Test resources

Maven Repositories

- Repositories where you can download Java libraries, maintained by community.
- Online repositories:
  - https://repo1.maven.org/maven2/ (Central)
  - https://packages.atlassian.com/mvn/maven-atlassian-external/
  - https://oss.sonatype.org/content/repositories/releases/
  - https://repo.hortonworks.com/content/repositories/releases/
  - https://repo.spring.io/plugins-release/
- Different repos have different amount of packages
  - The Central one being the largest

Project Object Model configuration – pom.xml

- An XML file that contains information about the project and configuration details used by Maven to build the project.
- Stores the project's dependencies.

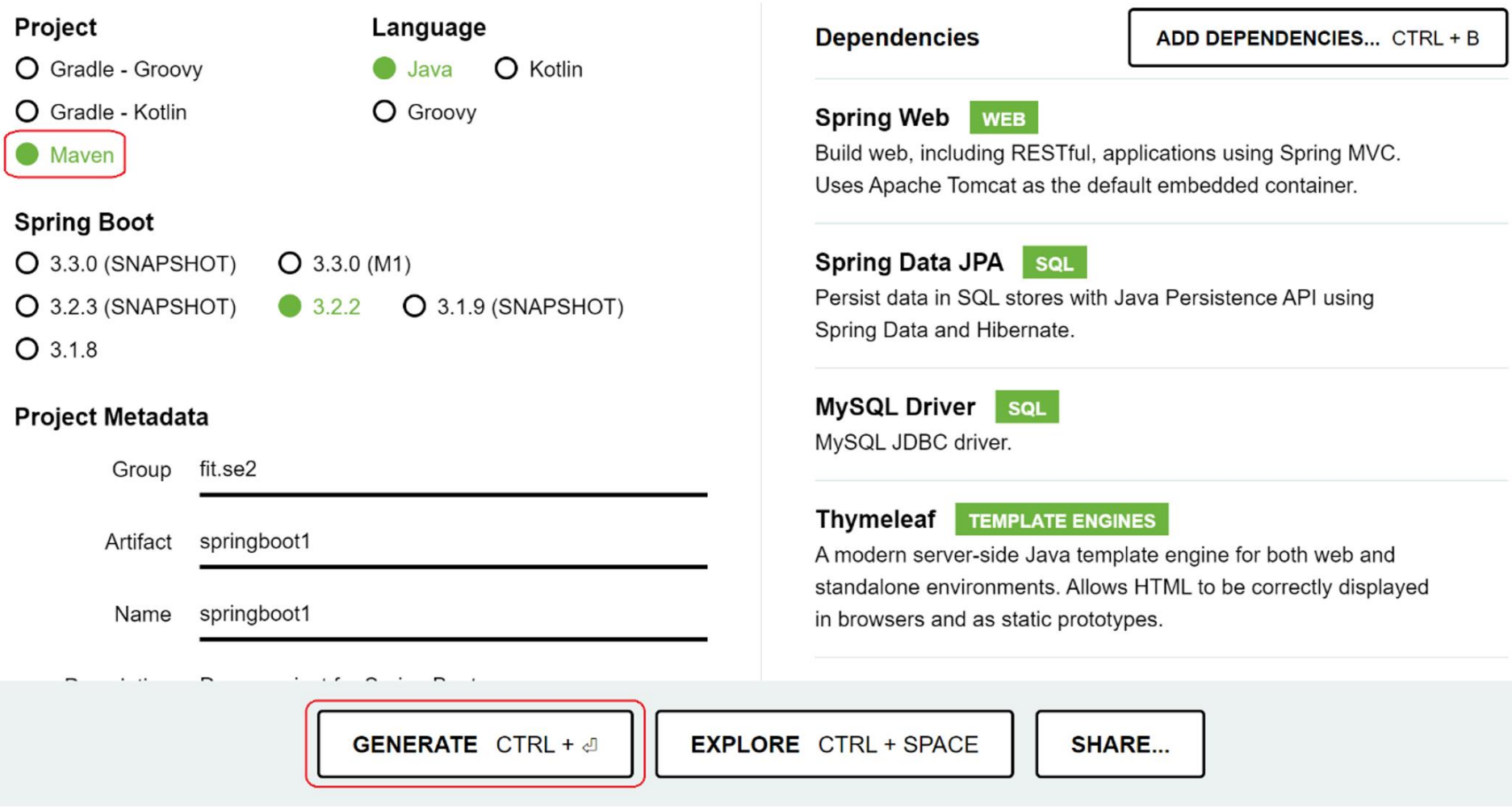
```
<dependencies>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
    </dependency>
    <dependency>
        <groupId>com.mysql</groupId>
        <artifactId>mysql-connector-j</artifactId>
        <scope>runtime</scope>
    </dependency>

</dependencies>
```

Introduction to Spring Boot

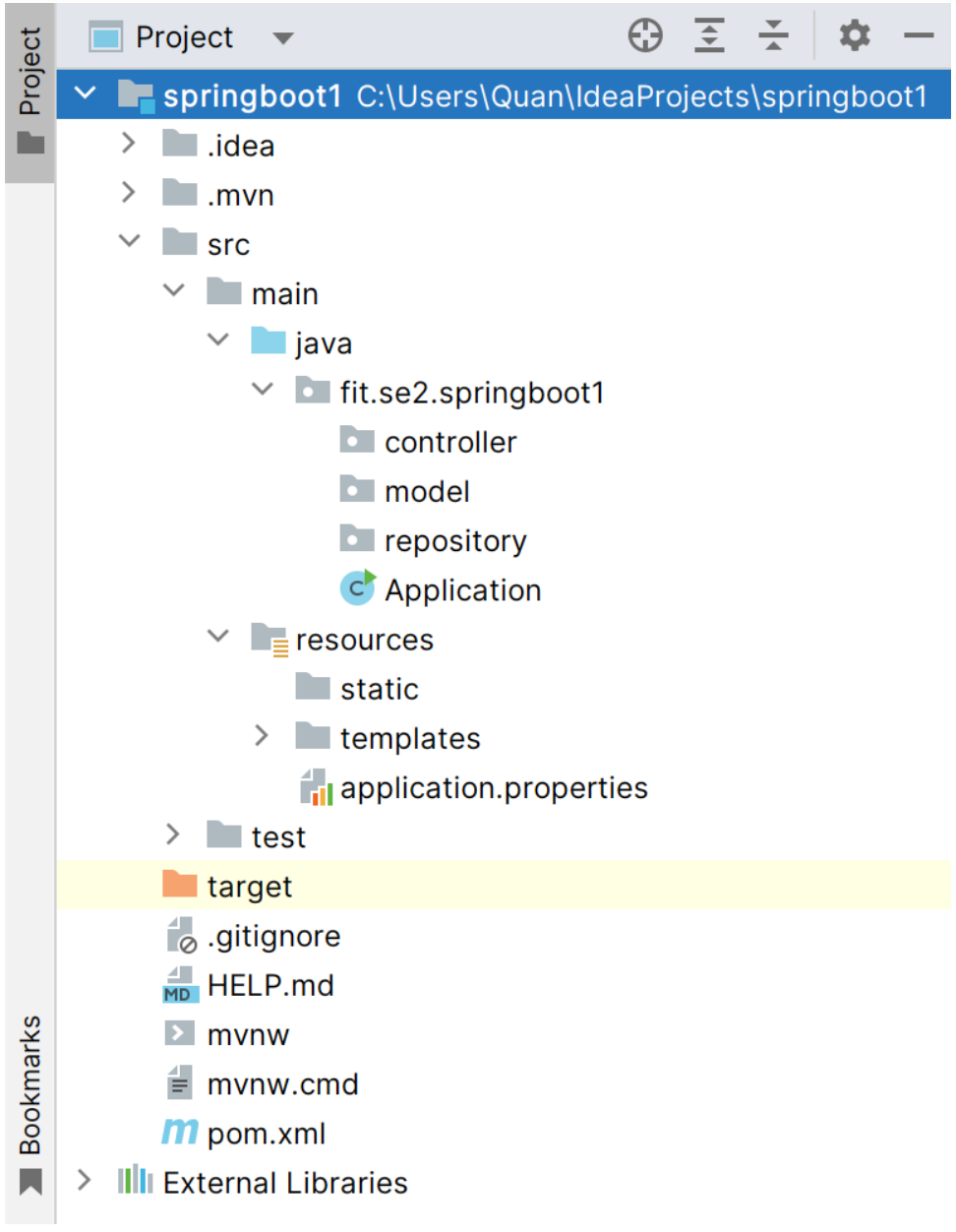
What are Spring and Spring Boot?

- Spring is the most popular, modern application development framework for enterprise Java.
  - Spring provides the infrastructures so that developers can focus on application-level business logic.
  - The goal: developers don't have to solve problems such as coupling, portability, etc.
- Spring Boot makes it easy to create stand-alone, production-grade Spring based Applications.
  - There are many ways to create a Spring Boot project, one of them is to visit: https://start.spring.io/
  - Spring Boot supports Java 17+

Creating Spring Boot project



Spring Boot project structure



Spring Boot `application.properties`

- A key configuration file in a Spring Boot project.
  - Located at: `src/main/resources/application.properties`
  - Used to configure various aspects of the application: DB connections, server settings, logging, security…
- Properties defined here can be accessed from the application
  - By using the `@Value` annotation
  - By injecting the `Environment` object
- Example configurations:
  - `server.port=8080` – specifies the website's port.
  - `spring.application.name=userservice` – specifies the application's name to be "userservice".

Thymeleaf

- Thymeleaf is a modern server-side Java template engine for both web and standalone environments
  - We're using Thymeleaf for web
  - Thymeleaf works similarly to EJS, Pug, HandleBars, Twig, etc.
  - Thymeleaf templates still look and work like HTML, so they keep being useful design artifacts (for other purposes).

```
<table>
  <thead>
    <tr>
      <th th:text="#{msgs.headers.name}">Name</th>
      <th th:text="#{msgs.headers.price}">Price</th>
    </tr>
  </thead>
  <tbody>
    <tr th:each="prod: ${allProducts}">
      <td th:text="${prod.name}">Oranges</td>
      <td th:text="${#numbers.formatDecimal(prod.price, 1, 2)}">0.99</td>
    </tr>
  </tbody>
</table>
```

A few words about Thymeleaf

- Thymeleaf templates are stored at:

  `src/main/resources/templates`

- Thymeleaf templates has `.html` extensions
- Thymeleaf allows you to define your own dialect
  - It comes with a standard dialect “out-of-the-box”
- The standard dialect defines the special attributes starting with `th`
  - Examples: `th:text`, `th:action`, `th:value`, `th:each`, `th:if`

Spring Data JPA

Object Relational Mapping

- Object Relational Mapping (ORM) is a technique used in creating a *bridge* between object-oriented programs and (in most cases) relational databases
- The idea is to use OOP methods to do CRUD instead of SQL queries
- Instead of querying like this:

```
"SELECT id, name, email, country, phone_number FROM users WHERE id = 20"
```

- We use this:

```
users.GetById(20)
```

JPA and Spring Data JPA

- The Java Persistence API provides Java developers with an object/relational mapping facility for managing relational data in Java applications
- JPA is a set of interfaces that defines how to persist data in Java applications
  - It cannot be used by itself
  - There are concrete implementations of JPA (e.g. Hibernate)
- Spring Data JPA is built upon JPA
  - It has all features of JPA, with additional convenient features

Configuring Data Source

- Your application needs some source of data.
  - E.g. a database like MySQL
- The concept of ORM is so that it is easy to switch between database systems.
  - This is often done by modifying configuration settings.
- Data Source configuration keys (in `application.properties`):
  - `spring.datasource.url=jdbc:mysql://localhost:3306/db_example`
  - `spring.datasource.username=springuser`
  - `spring.datasource.password=ThePassword`
  - `spring.datasource.driver-class-name=com.mysql.jdbc.Driver`
  - `spring.jpa.hibernate.ddl-auto=update`

Entities

- An **Entity** in Spring Boot is a class that maps to a database table.
  - An entity class is annotated with `@Entity`
  - It must have a primary key (annotated with `@Id`)
  - Entities are often placed in the `model` package

```
@Entity
public class User {
    @Id
    private Long id;
    private String name;
    private String email;

    // Getters and setters
}
```

Repositories

- Suppose you want to create a CRUD repository for the `Employee` table, you can use the `JpaRepository` interface:

```
public interface EmployeeRepository extends JpaRepository<Employee, Long> {
}
```

- Later, if you want to save an employee, you can get an instance of the repository and do this:

```
employeeRepository.save(employee);
```

Repositories

- Spring will create repository implementations automatically, at runtime, from the repository interface.
- Therefore, you can use these methods without having to implement them:

```
List<Employee> employees
    = employeeRepository.findByFirstName("John");
Employee emp1 = employeeRepository.getById(15);
```

Controllers

- A **Controller** is a class that handles incoming HTTP requests and returns responses.
  - Is annotated with `@Controller`
  - Controllers are often placed under the `controller` package
- The `@RequestMapping` , `@GetMapping`, `@PostMapping` annotations are used to map HTTP requests to handler methods.
- The controller returns the *name of a view*.
  - i.e. name of the Thymeleaf view

Controllers

```java
@Controller
@RequestMapping("/hello")
public class HelloController {
    @GetMapping
    public String sayHello() {
        // return the name of the view (e.g., hello.html)
        return "hello";
    }
}
```

Software Reuse

Software reuse

- In most engineering disciplines, systems are designed by composing existing components that have been used in other systems.
- Software engineering has been more focused on original development but it is now recognised that to achieve better software, more quickly and at lower cost, we need a design process that is based on systematic software reuse.
- There has been a major switch to reuse-based development over the past 10 years.

Reuse-based software engineering

- ✧ System reuse
  - Complete systems, which may include several application programs may be reused.
- ✧ Application reuse
  - An application may be reused either by incorporating it without change into other or by developing application families.
- ✧ Component reuse
  - Components of an application from sub-systems to single objects may be reused.
- ✧ Object and function reuse
  - Small-scale software components that implement a single well-defined object or function may be reused.

Benefits of software reuse

Benefit	Explanation
Accelerated development	Bringing a system to market as early as possible is often more important than overall development costs. Reusing software can speed up system production because both development and validation time may be reduced.
Effective use of specialists	Instead of doing the same work over and over again, application specialists can develop reusable software that encapsulates their knowledge.
Increased dependability	Reused software, which has been tried and tested in working systems, should be more dependable than new software. Its design and implementation faults should have been found and fixed.

Benefits of software reuse

Benefit	Explanation
Lower development costs	Development costs are proportional to the size of the software being developed. Reusing software means that fewer lines of code have to be written.
Reduced process risk	The cost of existing software is already known, whereas the costs of development are always a matter of judgment. This is an important factor for project management because it reduces the margin of error in project cost estimation. This is particularly true when relatively large software components such as subsystems are reused.
Standards compliance	Some standards, such as user interface standards, can be implemented as a set of reusable components. For example, if menus in a user interface are implemented using reusable components, all applications present the same menu formats to users. The use of standard user interfaces improves dependability because users make fewer mistakes when presented with a familiar interface.

Problems with reuse

Problem	Explanation
Creating, maintaining, and using a component library	Populating a reusable component library and ensuring the software developers can use this library can be expensive. Development processes have to be adapted to ensure that the library is used.
Finding, understanding, and adapting reusable components	Software components have to be discovered in a library, understood and, sometimes, adapted to work in a new environment. Engineers must be reasonably confident of finding a component in the library before they include a component search as part of their normal development process.
Increased maintenance costs	If the source code of a reused software system or component is not available then maintenance costs may be higher because the reused elements of the system may become increasingly incompatible with system changes.

Problems with reuse

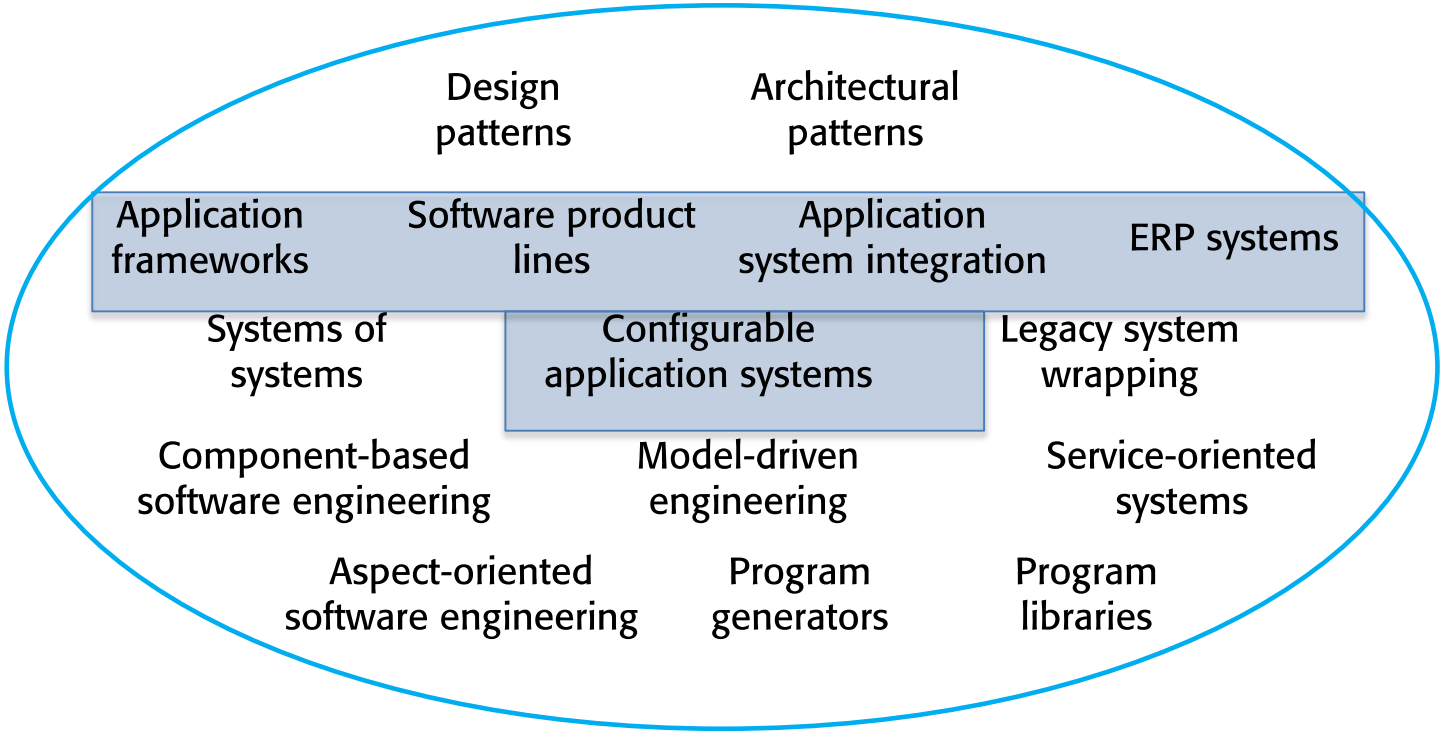
Problem	Explanation
Lack of tool support	Some software tools do not support development with reuse. It may be difficult or impossible to integrate these tools with a component library system. The software process assumed by these tools may not take reuse into account. This is particularly true for tools that support embedded systems engineering, less so for object-oriented development tools.
Not-invented-here syndrome	Some software engineers prefer to rewrite components because they believe they can improve on them. This is partly to do with trust and partly to do with the fact that writing original software is seen as more challenging than reusing other people's software.

The reuse landscape

The reuse landscape

- Although reuse is often simply thought of as the reuse of system components, there are many different approaches to reuse that may be used.
- Reuse is possible at a range of levels from simple functions to complete application systems.
- The reuse landscape covers the range of possible reuse techniques.

The reuse landscape



Approaches that support software reuse

Approach	Description
Application frameworks	Collections of abstract and concrete classes are adapted and extended to create application systems.
Application system integration	Two or more application systems are integrated to provide extended functionality
Architectural patterns	Standard software architectures that support common types of application system are used as the basis of applications. Described in Chapters 6, 11 and 17.
Aspect-oriented software development	Shared components are woven into an application at different places when the program is compiled. Described in web chapter 31.
Component-based software engineering	Systems are developed by integrating components (collections of objects) that conform to component-model standards. Described in Chapter 16.

Approaches that support software reuse

Approach	Description
Configurable application systems	Domain-specific systems are designed so that they can be configured to the needs of specific system customers.
Design patterns	Generic abstractions that occur across applications are represented as design patterns showing abstract and concrete objects and interactions. Described in Chapter 7.
ERP systems	Large-scale systems that encapsulate generic business functionality and rules are configured for an organization.
Legacy system wrapping	Legacy systems (Chapter 9) are ‘wrapped’ by defining a set of interfaces and providing access to these legacy systems through these interfaces.
Model-driven engineering	Software is represented as domain models and implementation independent models and code is generated from these models. Described in Chapter 5.

Approaches that support software reuse

Approach	Description
Program generators	A generator system embeds knowledge of a type of application and is used to generate systems in that domain from a user-supplied system model.
Program libraries	Class and function libraries that implement commonly used abstractions are available for reuse.
Service-oriented systems	Systems are developed by linking shared services, which may be externally provided. Described in Chapter 18.
Software product lines	An application type is generalized around a common architecture so that it can be adapted for different customers.
Systems of systems	Two or more distributed systems are integrated to create a new system. Described in Chapter 20.

Reuse planning factors

- The development schedule for the software.
- The expected software lifetime.
- The background, skills and experience of the development team.
- The criticality of the software and its non-functional requirements.
- The application domain.
- The execution platform for the software.

Application frameworks

Framework definition

- *"..an integrated set of software artefacts (such as classes, objects and components) that collaborate to provide a reusable architecture for a family of related applications."*

Application frameworks

- Frameworks are moderately large entities that can be reused. They are somewhere between system and component reuse.
- Frameworks are a sub-system design made up of a collection of abstract and concrete classes and the interfaces between them.
- The sub-system is implemented by adding components to fill in parts of the design and by instantiating the abstract classes in the framework.

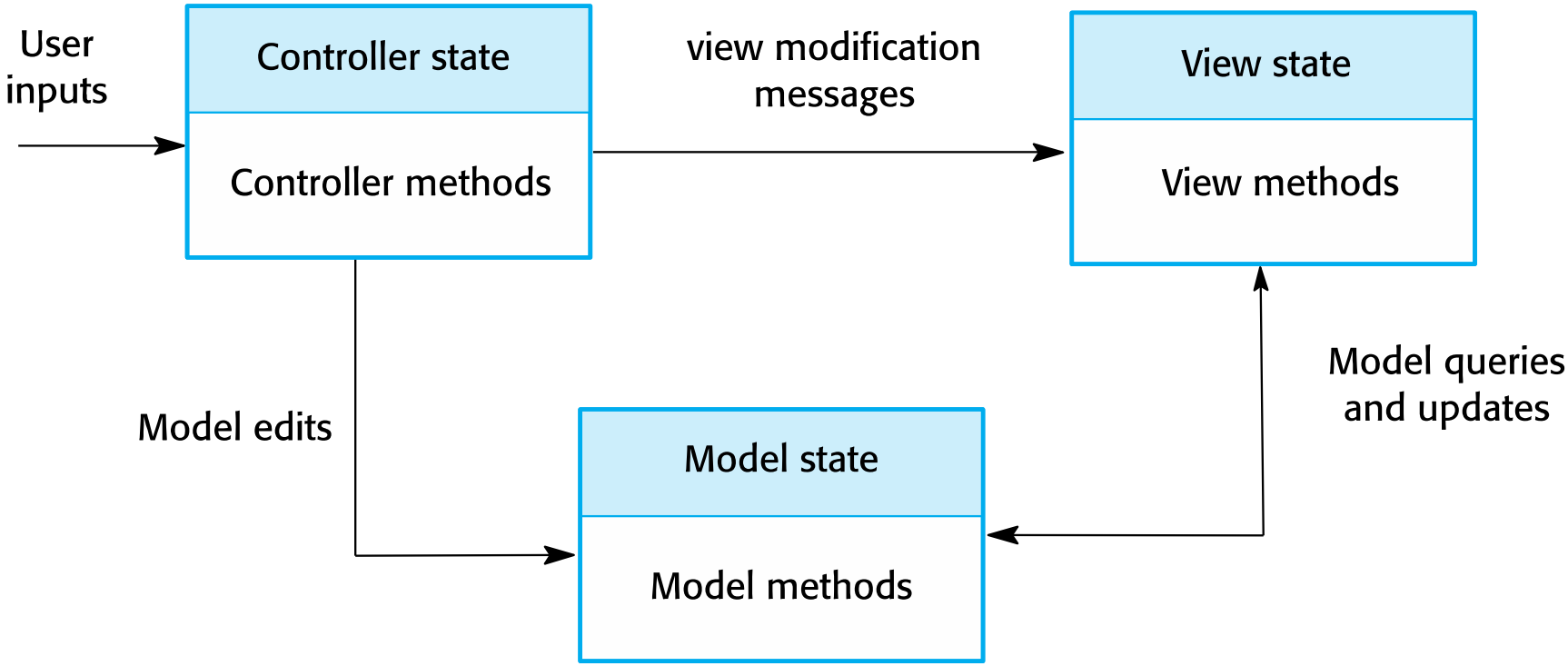
Web application frameworks

- Support the construction of dynamic websites as a front-end for web applications.
- WAFs are now available for all of the commonly used web programming languages e.g. Java, Python, Ruby, etc.
- Interaction model is based on the Model-View-Controller composite pattern.

Model-view controller

- ✧ System infrastructure framework for GUI design.
- ✧ Allows for multiple presentations of an object and separate interactions with these presentations.
- ✧ MVC framework involves the instantiation of a number of patterns.

The Model-View-Controller pattern



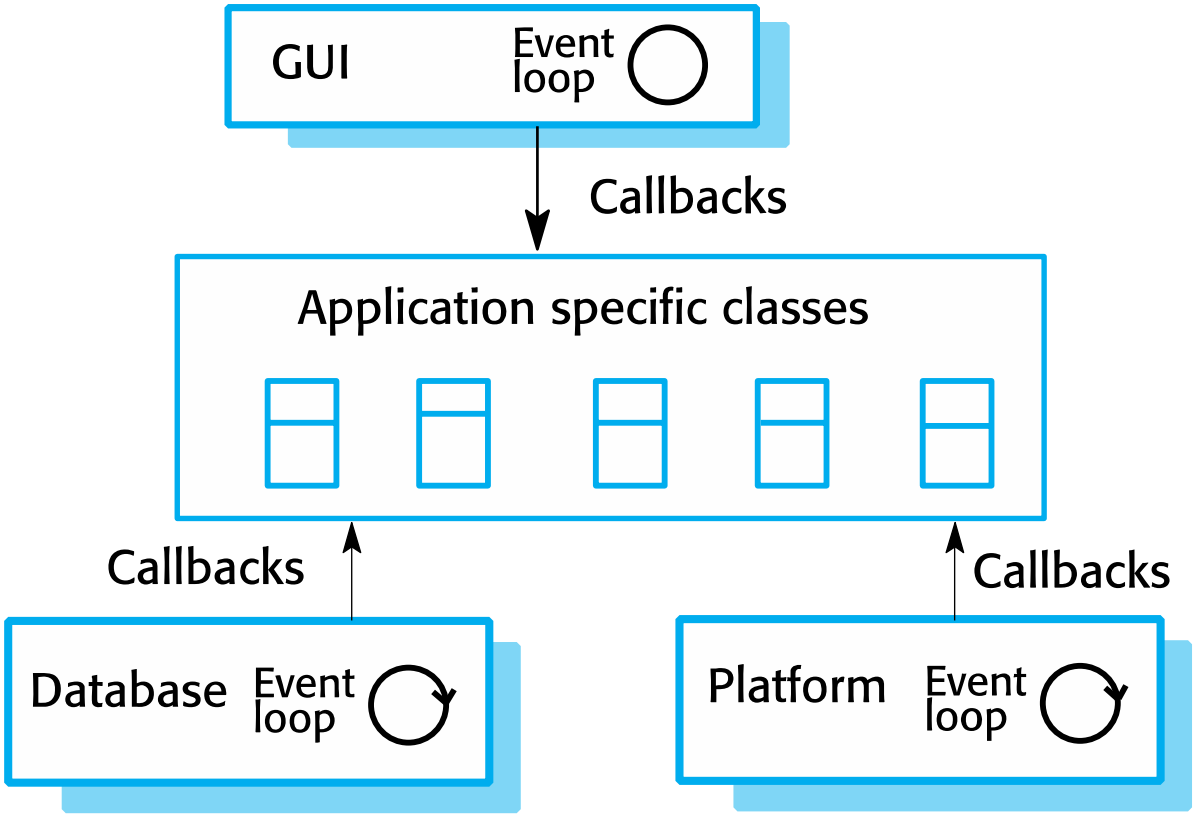
WAF features

- *Security*
  - WAFs may include classes to help implement user authentication (login) and access.
- *Dynamic web pages*
  - Classes are provided to help you define web page templates and to populate these dynamically from the system database.
- *Database support*
  - The framework may provide classes that provide an abstract interface to different databases.
- *Session management*
  - Classes to create and manage sessions (a number of interactions with the system by a user) are usually part of a WAF.
- *User interaction*
  - Most web frameworks now provide AJAX support (Holdener, 2008), which allows more interactive web pages to be created.

Extending frameworks

- Frameworks are generic and are extended to create a more specific application or sub-system. They provide a skeleton architecture for the system.
- Extending the framework involves
  - Adding concrete classes that inherit operations from abstract classes in the framework;
  - Adding methods that are called in response to events that are recognised by the framework.
- Problem with frameworks is their complexity which means that it takes a long time to use them effectively.

Inversion of control in frameworks



Framework classes

- ✧ System infrastructure frameworks
  - Support the development of system infrastructures such as communications, user interfaces and compilers.
- ✧ Middleware integration frameworks
  - Standards and classes that support component communication and information exchange.
- ✧ Enterprise application frameworks
  - Support the development of specific types of application such as telecommunications or financial systems.

Key points

- There are many different ways to reuse software. These range from the reuse of classes and methods in libraries to the reuse of complete application systems.
- The advantages of software reuse are lower costs, faster software development and lower risks. System dependability is increased. Specialists can be used more effectively by concentrating their expertise on the design of reusable components.
- Application frameworks are collections of concrete and abstract objects that are designed for reuse through specialization and the addition of new objects. They usually incorporate good design practice through design patterns.