Object Oriented Application Frameworks

Roadmap

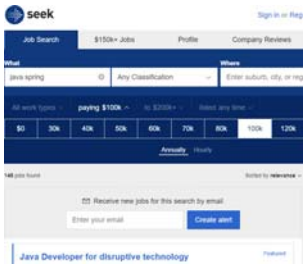
- Web Application Frameworks (WAFs)
- Spring (Java)
- Hibernate (Object Relational Mapping)
- Maven (Building software, describe dependencies)
- GIT (Version control)
- API design and integration
- Software engineering topics

- Lecture: concept and big picture about WAFs
- Labs: specifics about implementing and using WAFs

Learning Outcomes

- Understand how to build large scale reusable systems using Java frameworks
- Implement an application using Spring Framework and Hibernate Object relational Mapping
- Proficiently develop web applications using Eclipse, Maven and Version Control Systems (e.g. Git)
- Be able to compare Web Application frameworks
  - (choose based on architectural requirements)

Why Study OOAF?



Requirements

- Come to lectures and labs – expect 10 hrs/week
- Read materials provided
- Finish the project

- Bring your own laptop

Assessment Marking

- Proposal (10%): 2,000 word document. Group mark.
- Mid-term exam (30%). Concepts and programming languages used in the project. (individual)
- Project (40%): Implementation and demo. Group mark, scaled depending on individual contribution
- Final report (10%): Group mark plus individual contribution, scaled depending on individual contribution
- Presentation (10%) – YouTube video summarising outcomes of project

Assessment Standards

- Pass: Inconsistent programming and communication skills. Motivation and initiative. Must be able to build a complete web application using Spring and some API (e.g. Google)
- Credit: Show consistent programming and communication skills. Deliver a working application, quality documentation. Proficient coding and use of development
- Distinction: High quality programming and communication skills. Compare WAFs through analysis of advantages and disadvantages. Be able to relate pros and cons to a business problem. Experiment with frameworks to show how they can be exploited to a particular business problem.
- High-distintion: Exceleent programming and communication skills. Show independent work, ability to use and integrate several services and applications. Build interesting multi-service application Show high quality

Assessment Standards

- You must show your understanding of the whole system in the project (front-end to back-end)

Learning resources

- Tutors
- Canvas
- Spring, Hibernate and Google Code websites
- Software development tools (e.g. Eclipse, Git, Bitbucket etc.)
- Project description/templates

Put it all together

- This UoS aims to be very practical. You will be assessed on what you can deliver (ideas, code and presentation)
- Lectures will have basic descriptions of concepts
- Labs will be where you spend most of the time (you will be expected to do further lab work from home)
- Projects will require you to apply knowledge and skills from this course
- Assessment all during semester

Why use Object Oriented Application Frameworks

- Computing power has grown dramatically
  - Trade simplicity for power
- Software design is expensive and error-prone
  - Exacerbated by changes in hardware, software, OS and communication platform
- Many core concepts can be re-used

What is an Object Oriented Application Framework

- Framework is semi-complete application that can be customised
  - Re-use the body, re-write the code it calls
- Benefits:
  - Modularity
    - Encapsulate implementation details behind stable interfaces
      - Enhance reusability
    - Localise design and implementation changes
      - Easier to design and maintain
  - Extensibility
  - Dependency injection (inversion of control)
    - Objects given dependencies at creation time by some external entity instead of choosing which objects it collaborates with
    - Loose coupling: test and add objects independently
  - Reduced development time (with experience)

Weaknesses of Application Frameworks

- Extensible software is harder to develop
- Time to learn framework
- May require integration of multiple frameworks
- Must be able to adapt code to new frameworks
- Debugging may be harder
  - Hard to distinguish framework bugs from code bugs
  - Single stepping through code can be hard due to inversion of control
- Performance degradation

Frameworks only help with good code

- Frameworks are simply a form of design re-use
- Ideally you should also make use of:
  - Components (re-usable code)
  - Design patterns (regular forms of software code)

Design Patterns

- "you can use this solution a million times over, without ever doing it the same way twice"
- Maximising re-use requires anticipating new requirements and changes, designing accordingly

Design Patterns

- Common causes of re-design:
  - Creating an object by specifying a class explicitly
  - Dependence on specific operation
  - Dependence on hardware and software platform
  - Dependence on object representations or implementations
  - Algorithmic dependencies
  - Tight coupling.
  - Extending functionality by subclassing
  - Inability to alter classes conveniently
- Design patterns help ensure a system can change in specific ways
  - Make system independent of how objects created, composed and represented

Design Patterns

- Description of communicating objects and classes to solve a problem
- Classifies re-usable design structure:
  - Identify participating classes, instances, roles, collaborations and distribution of responsibilities
  - Often includes sample code
- Somewhat language dependent
  - Patterns rely on language features

Design Patterns

- Consist of:
  - Pattern name
  - Problem: describes when to apply pattern
  - Solution: elements, their relationships, responsibilities and collaborations
    - Abstract description and general arrangement of classes/objects
  - Consequences: typically space/time trade-offs
    - Impact on flexibility, extensibility and portability

Design Patterns

- Types:
  - Creational: How to create objects
  - Structural: How to construct a system with objects/components
  - Behavioural: How objects interact at run-time
- Look them up yourself!

Design Pattern Example

- Factory Builder (largely taken from journaldev.com)

```
public abstract class Computer {
  public abstract String getRAM();
  public abstract String getHDD();
  public abstract String getCPU();

  @Override public String toString(){
    return "RAM= "+this.getRAM()+",HDD="
        +this.getHDD()+", CPU="+this.getCPU(); }
}
```

```
public class PC extends Computer {
  private String ram;
  private String hdd;
  private String cpu;

  public PC(String ram, String hdd, String cpu){
    this.ram=ram;
    this.hdd=hdd;
    this.cpu=cpu; }

  @Override
  public String getRAM() {
    return this.ram; }
  @Override
  public String getHDD() {
    return this.hdd; }
  @Override
  public String getCPU() {
    return this.cpu; }
}
```

```
public class Server extends Computer {
  private String ram;
  private String hdd;
  private String cpu;

  /* stuff to load config file */
  public Server {
    this.ram=this.configFile.getProperty(ram);
    this.hdd=this.configFile.getProperty(hdd);
    this.cpu=this.configFile.getProperty(cpu); }

  @Override
  public String getRAM() {
    return this.ram; }
  @Override
  public String getHDD() {
    return this.hdd; }
  @Override
  public String getCPU() {
    return this.cpu; }
}
```

```
import design.model.Computer;
import design.model.PC;
import com.journaldev.design.model.Server;

public class ComputerFactory {
  public static Computer getComputer(String type,
                String ram, String hdd, String cpu){
    if("PC".equalsIgnoreCase(type))
      return new PC(ram, hdd, cpu);
    else if("Server".equalsIgnoreCase(type))
      return new Server();
    return null;
  }
}
```

```
package design.test;
import design.factory.ComputerFactory; import design.model.Computer;

public class TestFactory {

public static void main(String[] args) {

  System.out.print( "Do you want a Server? y/n" );
  String input = scanner.nextLine();
  if (input.equalsIgnoreCase("y"))
    Computer comp = ComputerFactory.getComputer("server")
  else {
    System.out.print( "Enter HDD for custom PC?" );
    String my_hhd = scanner.nextLine();
    System.out.print( "Enter RAM for custom PC?" );
    String my_RAM = scanner.nextLine();
    System.out.print( "Enter CPU speed for custom PC?" );
    String my_cpu = scanner.nextLine();
    Computer comp = ComputerFactory.getComputer("pc", my_hhd, my_RAM,
                                                my_cpu);
  }

  System.out.println("chosen_computer::"+ comp.toString());
  }
}
```

Benefits of Factory Pattern

- Abstraction
- Loosely coupled code
- Moves burden of creation from client to factory
- Easily extensible

Factory pattern encourages good coding

- Rule of thumb: Program to an interface, not an implementation
  - (don't only use inheritance for re-use)
    - Inheritance can offer identical interfaces for families of objects
      - Clients remain unaware of types of objects, so long as they adhere to an interface
      - Clients remain unaware of how the objects are implemented
        - Reduced implementation dependencies

Reading

- Mohamed Fayad and Douglas C. Schmidt. ``Object-oriented application frameworks``. Communications of the ACM. 40 (10). pp 32-38.
- Design Patterns: Elements of Reusble Object-Oriented Software. Gamma, Helm, Johnson and Vlissides. Introduction (Chapter 1) (Library)
- https://www.journaldev.com/1827/java-design-patterns-example-tutorial
- Wiki
- Random googling…