Feature Toggles

Hiding Work In Progress

- Feature Branches
  - develop each feature on a branch
  - merge to mainline when ready
- Feature Toggles
  - develop everything in mainline
  - disable/hide incomplete features

Feature Branching - Pros

- mainline protected during development
- no risk of accidental exposure

Feature Branching - Cons

- delayed integration
- merge conflicts
- refactoring avoidance
- onced merged hard to back out

Feature Toggles - Cons

- extra work during development
- more complicated to test
- might accidentally expose features

Feature Toggles - Pros

- continuous integration
- no merge conflicts
- increased visibility into all work streams
- better ability to refactor the code
- separate deployment from release
- enables continuous delivery

Simple Toggles in View

```
<c:if test="${featureFoo}">
  <a href="/foo">Foo</a>
</c:if
```

Simple Toggles in Code

```
public void doSomething() {
  if (featureFoo) {
     «foo specific logic»
  }
  «regular logic»
}
```

Spaghetti Toggles

```
public void doSomething() {

    if (featureFoo) {
        «foo specific logic»
    }

    «regular logic»

    if (featureFoo) {
        «more foo logic»

        if (featureBar) {
            «bar logic»
        }
    }
}
```

Maintainable Toggles

- Remember OO principles
- How do we extend existing behavior?

Maintainable Toggles

- Remember OO principles
- How do we extend existing behavior?

Maintainable Toggles

- Minimize conditionals
- Centralize toggle decisions
- Inheritance
- Composition
- Design patterns?

Extension Points

```
public interface Processor {
  void process(Bar bar);
}

public class CoreProcessor implements Processor {
   public void process(Bar bar) {
       doSomething(bar);
       handleFoo(bar);
       doSomethingElse(bar);
   }
   protected void handleFoo(Bar bar) { }
}

public class FooProcessor extends CoreProcessor {
    protected void handleFoo(Bar bar) {
        doSomethingFooSpecific(bar);
    }
}
```

Composition

```
public class Processor {
    public Processor(FeatureHandler handler) {
        this.handler = handler;
    }

    public void process(Bar bar) {
      doSomething();
      handler.handle(bar);
      doSomethingElse();
    }
}

public class FooHandler implements Handler {
    public void handle(Bar bar) {
        doSomethingCompletelyDifferent(bar);
    }
}
```

Maintainable Toggles

- Minimize conditionals
- Centralize toggle decisions
- Leverage Dependency Injection

Handling static assets

- How can we toggle CSS?
- How can we toggle JS?

Handling static assets

- How can we toggle CSS?
- How can we toggle JS?
- What about leaking information?

Handling Static Assets

- Turn JS/CSS into templates, run-time
  - consider interactions with CDN
- Render JS/CSS templates during build
  - consider ability to toggle at run-time
- Create feature specific JS/CSS
  - include them conditionally in the view

When do we toggle?

- Toggle at build-time

When do we toggle?

- Toggle at build-time
- Toggle at application startup

When do we toggle?

- Toggle at build-time
- Toggle at application startup
- Toggle during runtime

Things to consider

- Ensure only tested configurations go live
- Turn around time for making changes

Runtime toggles

- Persist toggles across application restarts?
- Change toggles across application servers
- Consider in-flight sessions

Final thoughts

- Remove toggles once code stabilizes
  - add something to the backlog
- Testing combinations
  - only need to test expected combinations