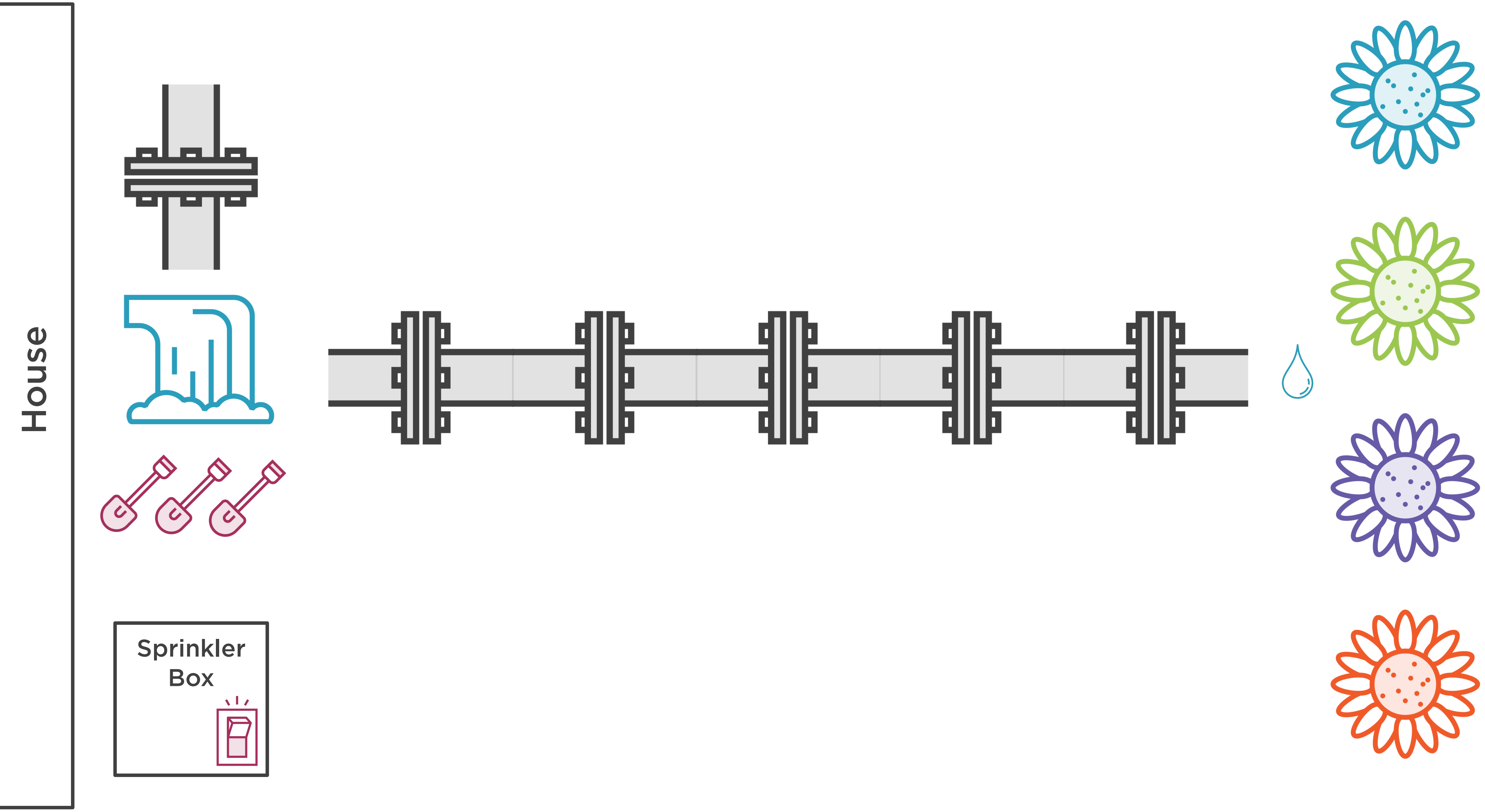
Secure Coding Practices in Java Applications (Java SE 11 Developer Certification 1Z0-819)

DESIGNING SECURE CODE

Josh Cummings
PRINCIPAL SOFTWARE ENGINEER

@jzheaux blog.jzheaux.io


House
Sprinkler Box

The earlier you can find security bugs, the fewer breaches you’ll have

Java 11 Certification Exam - Security

Oracle Secure Coding Guide

https://bit.ly/oracle-secure-coding

Simplify Your Code

C. A. R. Hoare said:

```
int sumFirstNValues(int[] values, int n) {
    return IntStream.of(values).limit(n).sum();
}
```

...there are obviously no deficiencies...

```
int sum(int[] v, int c){
    int j=0;
    for (int i=0;i<c;i++){
        j+=v[i];
    }
    return j;
}
```

...there are no obvious deficiencies...

Avoid Duplication

Minimize Permission Checks

```
@PreAuthorize(
"hasRole('ADMIN') || " +
"authentication.subscription == 'premium' && " +
"authentication.groups.contains('lib2')")
```

- Re-evaluated every time
- Hard to read
- Hard to test

```
@PreAuthorize("@authz.authorize(#root)")
```

- Re-evaluated every time
- Easy to read, though obscure meaning

```
@PreAuthorize("hasAuthority('file.share')")
```

- Evaluated once on login
- Intuitive authority name

Document Security

```
/**
* Impersonate {@code toBeImpersonated}. Once this method
* is successfully invoked, the system will consider {@code toBeImpersonated}
* to be logged in, which means that all operations will be done with
* the permission level of {@code toBeImpersonated}.
*
* Note that the {@code impersonator} can still be queried by calling
* {@code ImpersonatedUser#getImpersonator}.
*
* It is always true that {@code impersonator} must have
* <a href="https://docs.example.org/authz/privileges">higher privileges</a> than
* {@code toBeImpersonated} for this method to succeed.
*
* Both successful and failed impersonations are logged, along with reasons
* for the decision.
*
* @returns the {@code ImpersonatedUser}, which delegates all calls down to
* {@code toBeImpersonated} and also maintains a reference to {@code impersonator}
*/
public ImpersonatedUser impersonate(User impersonator, User toBeImpersonated) {
    // …
}
```

Secure Third-party Code

Keep dependencies up-to-date

Consider the maintenance impact of each library

* Keep Code Simple

* Avoid Duplication

* Minimize Permission Checks

* Document Security

* Secure Third-party Code

```java
public class Person implements Cloneable {

    // ...

    public Person clone() throws CloneNotSupportedException {
        return (Person) super.clone();
    }

}
```

Using Cloneable

Java's copy constructor

Implement Cloneable **and override** Object#clone

Now, person.clone() **will create a shallow copy**

```java
public interface Cloneable {
    // nothing to do?
}
```

Avoid Cloneable

Using implements **misleads that there's nothing more to do**

Breaks encapsulation by bypassing the constructor

```
public interface Serializable {
    // nothing to do?
}
```

Avoid **Serializable**

Breaks encapsulation by bypassing the constructor

Using implements **misleads that there's nothing more to do**

Secure Serialization

Call Constructor

By overriding *readResolve*

Opt-out

By overriding *readObject* and *writeObject*

Configure Allowlist

By using *ObjectInputFilter*

Secure Objects

In general, remember to:

- Keep Code Simple
- Avoid Duplication
- Minimize Permission Checks
- Detail Security
- Secure Third-party Code

When designing objects remember:

- Encapsulation
- Immutability
- Input Validation

Avoid Cloneable **and** Serializable