COSC345

Introduction to Refactoring (B)

Extract subclass

- A class has some fields or methods that are not always applicable or useful.
- So create a new subclass and move those fields and/or methods there.
- Example: not all InputStreams can support a #peek method, so create a PeekableInputStream (abstract) class, and move #peek and everything that depends on it there.

Extract superclass

- You have two or more classes with a lot in common.
- So create a new superclass for them, and move as much common stuff as you can up into it.
- Example: Bags and Sets are very similar, so create AbstractSetLikeCollection to be their parent. (In my library it has 66 instance methods and 63 class methods.)

Extract superclass and sibling

- You have a class with some conditional behaviours. The condition does not change after creation.
- So create a new superclass, move the unconditional methods there, and a new sibling, with one code for one condition moving to one subclass, and for the other to the other.
- Example: Heap -> AbstractHeap [Heap, BoundedHeap]

Merge only child

- A parent class has only one child and you do not expect more.
- So merge the child into its parent.
- This is the opposite of extract superclass or extract subclass.

Encapsulate field.

- A field is mutable.
- So ensure that it cannot be accessed directly outside the class.
- If you want it to be readable outside the class, create a getter method.
- If you want it to be modifiable outside the class, think again. If you still want this, create a setter method, and include code to check that the new value is reasonable.

Remove dangerous setters

- If a field should not be changed after initialisation,
- ensure that no public methods can change it, in particular that there are no setters for it.
- In Java, use 'final' if possible.

Virtualise field

- Most instances of a class have the same value for a particular field.
- Replace that field by a static *weak* hash table mapping instances of the class to values for the field.
- obj.getField() => if fieldMap has-key obj then fieldMap 's-value-for obj else default-value.
-

Encapsulate method.

- If a method isn't intended to be part of the public interface of a class, make it as local as possible (**private** beats **protected** beats package beats **public**).
- Public methods may have to check their arguments; private methods should not need to.
- It is much safer to change private methods.

Combine similar methods

- You have two or more functions/methods that are almost the same.
- Create a new private function/method that expresses what they have in common, with extra parameter(s) to express the differences.
- Make the old functions/methods call the new one.
- Classic example: write to a file and convert to a stream don't different in what to generate, just where to send it.

Pull method up

- You have subclass methods that do the same thing.
- So pull one copy up into the superclass and delete the others.
- If a class has some subclasses with such a method and others without, is it missing from the ones without? Is it harmless to add to them?

Push method down

- A class has a method.
- It makes sense for some of its subclasses but not all.
- So push it down into the ones where it does make sense.
- That makes copies; should the copies be different in their new homes (simpler?)
- Alternatively ...

Partition subclasses

- A class has several subclasses. Some of them have a property in common, the others don't.
- So make a new subclass, and make the ones with the common property subclasses of that instead.
- What if there are two incompatible properties?
- How does a support tool know what properties a class has?

Convert constructor to factory

- You have several constructors, but it is hard to tell them apart.
- Or you want a "constructor" that actually returns an instance of a subclass, or an existing instance.
- Ensure that you have one constructor, maybe private, that can initialise everything, and convert others to factory methods.