Software Patterns

• There is a large and growing literature & community centered around “software patterns”

• “Software patterns” are (frequently informally) identified interactions among software elements that serve some need

• Frequently the same sort of pattern can play a role in many different contexts
Motivations

• A rule is duplicated in many different forms in different code
  – Some to see if particular object matches (if or switch/case stmts.)
  – Some as SQL query
  – Some as test to collect from collection
  – Some as assertions

• Burying of rule abstraction in somewhat arbitrary object

• There is a combinatorial explosion of different methods for varying subsets of data
Bad Smell 1: Combinatorial Explosion

- Count Males/Females/young/elderly/First Nations/Metis/Caucasians/Male First Nations/Female First Nations/.../Susceptible/Infective/...Combinations...

- Compute prevalence among Males/Females/young/elderly/First Nations/Metis/Caucasians/Male First Nations/Female First Nations/.../Susceptible/Infective/...Combinations...

- Perform intervention on Males/Females/young/elderly/First Nations/Metis/Caucasians/Male First Nations/Female First Nations/.../Susceptible/Infective/...Combinations...
Bad Smell 2: Many Independent Statistics

• When called, each statistic performs a separate pass over the population
Key Idea

• Single immutable object that specifies some subset of domain objects
  – Often encapsulates general criteria
  – collects all related logic for a condition
    • Count agents matching condition
    • Count fraction of agents matching condition A & B out of those matching just A (e.g. fraction of Children that are infected)
    • Select objects (retrieves a list)
      – matching specific criteria
      – matching range of criteria
    • Specify conditions for who is acted upon (e.g. via intervention)
• Factory creates with enough context to evaluate predicate from simple arguments (e.g. agent)
Enhancing the Concept

• Subtyping of specifications
• Enabling combination of specifications
Specification Subtyping

**IPersonPredicate**

isSatisfiedBy(Person)

---

**SexPredicate**

SexPredicate (Sex sexToRecognize) isSatisfiedBy(Person)

**EthnicityPredicate**

EthnicityPredicate (Ethnicity ethnicityToRecognize) isSatisfiedBy(Person)

**AgePredicate**

AgePredicate (int ageMin, int ageMax) isSatisfiedBy(Person)
Counting Persons Matching a Specification
Method Body

```java
// despite slight performance hit, just delegate to machinery for matching multiple predicates
IPersonPredicate arrayPredicates[] = { predicate };
int [] arrayCountMatchingSpecification = countPeopleMatchingMultiplePredicates(arrayPredicates);

return arrayCountMatchingSpecification[0];
```
Method to Count Matching Multiple Specifications
int countSpecifications = arrayPredicates.length;
int[] arrayCountMatchingSpecification = new int[countSpecifications];

for (Person p : Population) {
    for (int i = 0; i < countSpecifications; i++) {
        if (arrayPredicates[i].isSatisfiedBy(p)) {
            arrayCountMatchingSpecification[i]++;
        }
    }
}

return arrayCountMatchingSpecification;
Function to Retrieve People Matching Specification
Body of Code

```java
    ArrayList<Person> accumulatedPeopleMatchingSpecifications = new ArrayList<Person>();
    for (Person p : Population) {
        if (spec.Predicate(p)) {
            accumulatedPeopleMatchingSpecifications.add(p);
        }
    }
    return accumulatedPeopleMatchingSpecifications;
```
Method to Perform Action Based on Specification
Performing Action

```java
int countMatchingSpecification = 0;

for (Person p : Population) {
    if (spec.Predicate(p)) {
        action.PerformAction(p);
        countMatchingSpecification++;
    }
}

return countMatchingSpecification;
```
Reporting List
Reporting Summary Statistics

Function body:

```java
trace("Summary statistics for time "+ time());

String[] statisticsNames = reportingStatistics.keySet().toArray();
IFramePredicate predicates[] = predicatesForNames(statisticsNames);
int[] countsForStatistics[] = countPeopleMatchingMultiplePredicates(predicates);

for (int i = 0; i < statisticsNames.length; i++)
    trace(" + statisticsNames[i] + ": " + countsForStatistics[i]);
```
Creating the Reporting List

```java
// Function: initializeSummaryStatistics

Function body:

IF PersonPredicate malePredicate = new SexPredicate(Person.Sex.Male);
reportingStatistics.put("Men", malePredicate);
IF PersonPredicate femalePredicate = new SexPredicate(Person.Sex.Female);
reportingStatistics.put("Women", femalePredicate);

reportingStatistics.put("Senior citizens", new SeniorCitizenPredicate());
reportingStatistics.put("Children", new ChildPredicate());

IF PersonPredicate firstNationsPredicate = new EthnicityPredicate(Person.Ethnicity.FirstNations);
reportingStatistics.put("First Nations", firstNationsPredicate);
IF PersonPredicate metisPredicate = new EthnicityPredicate(Person.Ethnicity.Metis);
reportingStatistics.put("Metis", metisPredicate);

reportingStatistics.put("First Nations Males", new AndPersonPredicate( malePredicate,
firstNationsPredicate));
reportingStatistics.put("First Nations Females", new AndPersonPredicate(femalePredicate,
firstNationsPredicate));
reportingStatistics.put("Metis Males", new AndPersonPredicate( malePredicate,
metisPredicate));
```