Principles of Software Construction: Objects, Design, and Concurrency

Software engineering in practice

Toward people and process

Josh Bloch  Charlie Garrod  Darya Melicher
Administrivia

- Homework 5c due last night!
- Homework 6 coming soon
  - Checkpoint deadline
- Happy Thanksgiving break!
Key concepts from last Thursday
# Lambda syntax

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>parameter -&gt; expression</td>
<td>(x \to x \times x)</td>
</tr>
<tr>
<td>parameter -&gt; block</td>
<td>(s \to {\text{System.out.println}(s);})</td>
</tr>
<tr>
<td>(parameters) -&gt; expression</td>
<td>((x, y) \to \text{Math.sqrt}(x \times x + y \times y))</td>
</tr>
</tbody>
</table>
| (parameters) -> block                        | \((s_1, s_2) \to \{
|                                           | \hspace{1em} \text{System.out.println}(s_1 + "," + s_2);\}\) |
| (parameter decls) -> expression             | \((\text{double } x, \text{double } y) \to \text{Math.sqrt}(x \times x + y \times y)\) |
| (parameter decls) -> block                   | \((\text{List}\langle ?\rangle \text{list}) \to \{
|                                           | \hspace{1em} \text{Arrays.shuffle(list); Arrays.sort(list);}\}\) |
Method references – a more succinct alternative to lambdas

• Lambdas are succinct
  
  \[
  \text{map.merge(key, 1, (count, incr) -> count + incr)};
  \]

• But *method references* can be more so

  \[
  \text{map.merge(key, 1, \text{Integer::sum})};
  \]

• The more parameters, the bigger the win
  – But parameter names *may* provide documentation
  – If you use a lambda, choose parameter names carefully!
Simple stream examples – mapping, filtering, sorting, etc.

List<String> longStrings = stringList.stream()
   .filter(s -> s.length() > 3)
   .collect(Collectors.toList());

List<String> firstLetters = stringList.stream()
   .map(s -> s.substring(0,1))
   .collect(Collectors.toList());

List<String> firstLettersOfLongStrings = stringList.stream()
   .filter(s -> s.length() > 3)
   .map(s -> s.substring(0,1))
   .collect(Collectors.toList());

List<String> sortedFirstLettersWithoutDups = stringList.stream()
   .map(s -> s.substring(0,1))
   .distinct()
   .sorted()
   .collect(Collectors.toList());
What Josh didn't show you...
public interface Stream<T> extends BaseStream<T, Stream<T>> {
    // Intermediate Operations
    Stream<T> filter(Predicate<T>);
    <R> Stream<R> map(Function<T, R>);
    IntStream mapToInt(ToIntFunction<T>);
    LongStream mapToLong(ToLongFunction<T>);
    DoubleStream mapToDouble(ToDoubleFunction<T>);
    <R> Stream<R> flatMap(Function<T, Stream<R>>);
    IntStream flatMapToInt(Function<T, IntStream>);
    LongStream flatMapToLong(Function<T, LongStream>);
    DoubleStream flatMapToDouble(Function<T, DoubleStream>);
    Stream<T> distinct();
    Stream<T> sorted();
    Stream<T> sorted(Comparator<T>);
    Stream<T> peek(Consumer<T>);
    Stream<T> limit(long);
    Stream<T> skip(long);
Stream interface is a monster (2/3)

// Terminal Operations
void forEach(Consumer<T>); // Ordered only for sequential streams
void forEachOrdered(Consumer<T>); // Ordered if encounter order exists
Object[] toArray();
<A> A[] toArray(IntFunction<A[]> arrayAllocator);
T reduce(T, BinaryOperator<T>);
Optional<T> reduce(BinaryOperator<T>);
<U> U reduce(U, BiFunction<U, T, U>, BinaryOperator<U>);
<R, A> R collect(Collector<T, A, R>); // Mutable Reduction Operation
<R> R collect(Supplier<R>, BiConsumer<R, T>, BiConsumer<R, R>);
Optional<T> min(Comparator<T>);
Optional<T> max(Comparator<T>);
long count();
boolean anyMatch(Predicate<T>);
boolean allMatch(Predicate<T>);
boolean noneMatch(Predicate<T>);
Optional<T> findFirst();
Optional<T> findAny();
Stream interface is a monster (3/3)

// Static methods: stream sources
public static <T> Stream.Builder<T> builder();
public static <T> Stream<T> empty();
public static <T> Stream<T> of(T);
public static <T> Stream<T> of(T...);
public static <T> Stream<T> iterate(T, UnaryOperator<T>);
public static <T> Stream<T> generate(Supplier<T>);
public static <T> Stream<T> concat(Stream<T>, Stream<T>);
In case your eyes aren’t glazed yet

public interface BaseStream<T, S extends BaseStream<T, S>>
    extends AutoCloseable {
    Iterator<T> iterator();
    Spliterator<T> spliterator();
    boolean isParallel();
    S sequential(); // May have little or no effect
    S parallel(); // May have little or no effect
    S unordered(); // Note asymmetry wrt sequential/parallel
    S onClose(Runnable);
    void close();
}
It keeps going: java.util.stream.Collectors

... toList()
... toMap(...)
... toSet(...)
... reducingBy(...)
... groupingBy(...)
... partitioningBy(...)
It keeps going: java.util.stream.Collectors

... toList()
... toMap(...)
... toSet(...)
... reducingBy(...)
... groupingBy(...)
... partitioningBy(...)
Optional<T> – a third way to indicate the absence of a result

public final class Optional<T> {
    boolean isPresent();
    T get();

    void ifPresent(Consumer<T>);
    Optional<T> filter(Predicate<T>);
    <U> Optional<U> map(Function<T, U>);
    <U> Optional<U> flatMap(Function<T, Optional<U>>);
    T orElse(T);
    T orElseGet(Supplier<T>);
    <X extends Throwable> T orElseThrow(Supplier<X>) throws X;
}
Changes to existing libraries... e.g.,

```java
public interface Collection<E> {
    ...
    default Stream<E> stream();
    default Stream<E> parallelStream();
    default Spliterator<E> spliterator();
}
```
Overall: Streams design discussion

• Recall the fundamental API design principles...
Today: Software engineering in practice

- An introduction to software engineering
- Methodologies discussion: Test-driven development
What is software engineering?
1968 NATO Conference on Software Engineering
Compare to other forms of engineering

• e.g., Producing a car or bridge
  – Estimable costs and risks
  – Well-defined expected results
  – High quality

• Separation between plan and production

• Simulation before construction

• Quality assurance through measurement

• Potential for automation
Software engineering is “the establishment and use of sound engineering principles in order to obtain, economically, software that is reliable and works efficiently on real machines.”

[Bauer 1975, S. 524]
Software engineering in the real world

- e.g., HealthCare.gov
  - Estimable costs and risks
  - Well-defined expected results
  - High quality
- Separation between plan and production
- Simulation before construction
- Quality assurance through measurement
- Potential for automation
Major topics in 17-313 (Foundations of SE)

- Process considerations for software development
- Requirements elicitation, documentation, and evaluation
- Design for quality attributes
- Strategies for quality assurance
- Empirical methods in software engineering
- Time and team management
- Economics of software development
The foundations of our Software Engineering program

• Core computer science fundamentals
• Building good software, organizing software projects
  – Development teams, customers, and users
  – Process, requirements, estimation, management, and methods
• The larger context of software
  – Business, society, policy
• Engineering experience
• Communication skills
  – Written and oral
Today: Software engineering in practice

- An introduction to software engineering
- Methodologies discussion: Test-driven development
Test-driven development (TDD), informally

Write a failing test → Make the test pass → Refactor

From Growing Object-Oriented Software by Nat Pryce and Steve Freeman
http://www.growing-object-oriented-software.com/figures.html

@sebrose  http://cucumber.io
Test-driven development rules

1. You may only write production code to make a failing test pass
2. You may only write a minimally failing unit test
3. You may only write minimal code to pass the failing test
Test-driven development as a design process

"The act of writing a unit test is more an act of design and documentation than of verification. It closes a remarkable number of feedback loops, the least of which pertains to verification."
Advantages of test-driven development

• Clear place to start
• Iterative, agile design process
• Less wasted effort?
• Robust test suite, including regression tests
A test-driven development demo: Diamond Kata

• Given a letter, generate a diamond starting at ‘A’, with the given letter at the widest point.
  – e.g., `diamond('C')` would generate:
    
    A
    B  B
    C  C
    B  B
    A
Test-driven development: Your impressions?
Empirical methods in software engineering

• How do we study the effectiveness of test-driven development compared to other methodologies?
Research on test-driven development (1/2)

- Hilton et al.: Students learn better when forced to write tests first
- Bhat et al.: At Microsoft, projects using TDD had greater than two times code quality, but 15% more upfront setup time
- George et al.: TDD passed 18% more test cases, but took 16% more time
- Scanniello et al.: Perceptions of TDD include: novices believe TDD improves productivity at the expense of internal quality
Research on test-driven development (2/2)

• Fucci et al.: Results: The Kruskal-Wallis tests did not show any significant difference between TDD and TLD in terms of testing effort (p-value = .27), external code quality (p-value = .82), and developers' productivity (p-value = .83).

• Fucci et al.: Conclusion: The claimed benefits of TDD may not be due to its distinctive test-first dynamic, but rather due to the fact that TDD-like processes encourage fine-grained, steady steps that improve focus and flow.
Summary

- Software engineering requires consideration of many issues, social and technical, above code-level considerations
- Interested? Take 17-313