Section A

Q1	10/10
Q2	6 /6
Q3	5 /5
Q4	8 /8
Q5	15/15
Q6	6 /6

Compilation penalty
Style penalty (capped at -3) /

Total for Section A 50 / 50

Username: jlk21

Compilation: 1 / 1

Model Answer's Tests – Question1Tests: 10 / 10

Model Answer's Tests – Question2Tests: 7 / 7

Model Answer's Tests – Question3Tests: 11 / 11

Model Answer's Tests – Question4Tests: 3 / 3

Model Answer's Tests – Question5Tests: 9 / 9

Model Answer's Tests – Question6Tests: 3 / 3

Model Answer's Tests – GoodPracticesTestsNote: 4 / 4

Model Answer's Tests – GoodPracticesTestsStandardTune: 2 / 2

Model Answer's Tests – GoodPracticesTestsTransposedTune: 2 / 2

Model Answer's Tests – GoodPracticesTestsTune: 1 / 1

No Google style violations – excellent!

Style penalty (capped at –3): 0

Note: if it is below the cap, your total style penalty could be higher if the marker has stylistic concerns that go beyond what Checkstyle identifies automatically.

```java
package tunes;

public final class Note {
  private final int pitch;
  private final int duration;

  private static final int MIN_PITCH = 0;
  private static final int MAX_PITCH = 200;

  private static final int MIN_DURATION = 1;
  private static final int MAX_DURATION = 64;

  private static final int NUM_NOTES_PER_OCTAVE = 12;

  public Note(int pitch, int duration) {
    if (pitch < MIN_PITCH || pitch > MAX_PITCH) {
      throw new IllegalArgumentException("Pitch is out of bounds.");
    }
    if (duration < MIN_DURATION || duration > MAX_DURATION) {
      throw new IllegalArgumentException("Duration is out of bounds.");
    }
    this.pitch = pitch;
    this.duration = duration;
  }

  public boolean hasNoteAbove() {
    return pitch < MAX_PITCH;
  }

  public boolean hasNoteBelow() {
    return pitch > MIN_PITCH;
  }

  public Note noteAbove() {
    // PRE: There is a note above
    return new Note(pitch + 1, duration);
  }

  public Note noteBelow() {
    // PRE: There is a note below
    return new Note(pitch - 1, duration);
  }

  public int getDuration() {
    return duration;
  }

  @Override
  public String toString() {
    int value = pitch % NUM_NOTES_PER_OCTAVE;
    String valueName = switch (value) {
      case 0 -> "C";
      case 1 -> "C#";
      case 2 -> "D";
      case 3 -> "D#";
      case 4 -> "E";
      case 5 -> "F";
      case 6 -> "F#";
      case 7 -> "G";
      case 8 -> "G#";
      case 9 -> "A";
      case 10 -> "A#";
      case 11 -> "B";
      default -> throw new RuntimeException(
          "Value out of bounds; check NUM_NOTES_PER_OCTAVE definition.");
    };
    int octave = pitch / NUM_NOTES_PER_OCTAVE;
    StringBuilder sb = new StringBuilder();
    sb.append(valueName);
    sb.append(octave);
    sb.append("(");
    sb.append(duration);
    sb.append(")");
    return sb.toString();
  }

  @Override
  public boolean equals(Object that) {
    if (!(that instanceof Note thatNote)) {
      return false;
    }
```



```java
    return thatNote.pitch == pitch && thatNote.duration == duration;
  }

  @Override
  public int hashCode() {
    return pitch * duration;
  }
}
```

```java
package tunes;

import java.util.ArrayList;
import java.util.List;

public class StandardTune implements Tune {
  private final List<Note> notes;

  public StandardTune() {
    notes = new ArrayList<>();
  }

  @Override
  public List<Note> getNotes() {
    return new ArrayList<>(notes);
  }

  @Override
  public void addNote(Note note) {
    notes.add(note);
  }
}
```



```java
package tunes;

import java.util.List;

public class TransposedTune implements Tune {
  private final Tune targetTune;
  private final int pitchOffset;

  public TransposedTune(Tune targetTune, int pitchOffset) {
    this.targetTune = targetTune;
    this.pitchOffset = pitchOffset;
  }

  @Override
  public List<Note> getNotes() {
    return targetTune
        .getNotes()
        .stream()
        .map(note -> shift(note, pitchOffset))
        .toList();
  }

  @Override
  public void addNote(Note note) {
    targetTune.addNote(shift(note, -pitchOffset));
  }

  private Note shift(Note note, int offset) {
    if (offset > 0 && note.hasNoteAbove()) {
      return shift(note.noteAbove(), offset - 1);
    } else if (offset < 0 && note.hasNoteBelow()) {
      return shift(note.noteBelow(), offset + 1);
    } else {
      return note;
    }
  }
}
```

good for avoiding duplicate code.

```
 1: package tunes;
 2:
 3: import java.util.List;
 4:
 5: public interface Tune {
 6:   List<Note> getNotes();
 7:
 8:   void addNote(Note note);
 9:
10:   default int getTotalDuration() {
11:     return getNotes()
12:         .stream()
13:         .mapToInt(Note::getDuration)
14:         .reduce(0, Integer::sum);
15:   }
16: }
```

```
Model Answer's Tests - Question1Tests works!

JUnit version 4.12
..........
Time: 0.004

OK (10 tests)


Model Answer's Tests - Question2Tests works!

JUnit version 4.12
.......
Time: 0.015

OK (7 tests)


Model Answer's Tests - Question3Tests works!

JUnit version 4.12
...........
Time: 0.016

OK (11 tests)


Model Answer's Tests - Question4Tests works!

JUnit version 4.12
...
Time: 0.016

OK (3 tests)


Model Answer's Tests - Question5Tests works!

JUnit version 4.12
.........
Time: 0.033

OK (9 tests)


Model Answer's Tests - Question6Tests works!

JUnit version 4.12
...
Time: 0.02

OK (3 tests)


Model Answer's Tests - GoodPracticesTestsNote works!

JUnit version 4.12
....
Time: 0.015

OK (4 tests)


Model Answer's Tests - GoodPracticesTestsStandardTune works!

JUnit version 4.12
..
Time: 0.016

OK (2 tests)


Model Answer's Tests - GoodPracticesTestsTransposedTune works!

JUnit version 4.12
..
Time: 0.016

OK (2 tests)

```



```
Model Answer's Tests - GoodPracticesTestsTune works!

JUnit version 4.12
.
Time: 0.014

OK (1 test)

```