

# Code Comprehension Tasks

Thank you for continuing with this study on how developers read and evaluate short pieces of Java code.

In this section, you will review **1+10 independent code snippets**, each containing a small bug or logic issue.

For each snippet, you will be asked to **understand what the code is doing** and **select or reason about a possible fix**.

Please complete each task in one sitting. Feel free to take breaks between tasks.

You may not run the code; all answers should be based on your own reasoning.

There are **no right or wrong answers** for most questions, only your best judgment.

Estimated time: **30–40 minutes**.

Your responses will remain anonymous and confidential.

\* Indicates required question

---

## 1. Consent to Participate

Please read the following information before beginning this study.

You are invited to participate in a research study about how developers understand and evaluate Java code.

In this part of the study, you will be asked to complete a short series of code comprehension tasks.

Your participation is **voluntary**, and you may stop at any time without penalty.

No personally identifying information will be collected, and your responses will remain **anonymous and confidential**.

Your data will be analyzed only in aggregate to better understand general programming and reasoning patterns.

If you have questions about the study, you may contact the research team at [g7shi@uwaterloo.ca](mailto:g7shi@uwaterloo.ca).

*Check all that apply.*

I have read the above information and agree to participate in this study.

## Note (This will NOT be in the final version)

These are **example tasks**. In other words, this will not be the final version of the experiment. Tasks 1 to 4 are tasks **without** AoCs, and tasks 5 to 8 are tasks **with** AoCs. The AoCs in those tasks are respectively:

- Conditional Operator,
- Infix Operator Precedence,
- Pre Increment/Decrement, and
- Type Conversion.

In the final version, we will **NOT** be using a Google Form. Instead, we will build a web app where the orders of the tasks and repair options are **randomized**. Additionally, the code snippets will be formatted so that they are easier to read.

### Task 1 Part 1

This Java method finds the largest number in an array.

```
public class BaselineExample {  
    public static int findMax(int[] numbers) {  
        if (numbers == null || numbers.length == 0) return 0;  
        int max = 0;  
        for (int n : numbers) {  
            if (n > max) {  
                max = n;  
            }  
        }  
        return max;  
    }  
  
    public static void main(String[] args) {  
        int[] values = {5, -3, 9, -7, 2};  
        System.out.println(findMax(values));  
    }  
}
```

2. Please explain what the bug is in 1-2 sentences. \*

---

---

---

---

---

**Task 1 Part 2 — Please inspect all three repair options before making your decision!**

This Java method finds the largest number in an array.

```
public class BaselineExample {  
    public static int findMax(int[] numbers) {  
        if (numbers == null || numbers.length == 0) return 0;  
        int max = 0;  
        for (int n : numbers) {  
            if (n > max) {  
                max = n;  
            }  
        }  
        return max;  
    }  
  
    public static void main(String[] args) {  
        int[] values = {5, -3, 9, -7, 2};  
        System.out.println(findMax(values));  
    }  
}
```

3. Repair option - Please rank this fix by the four criteria. Please DO NOT assign the same rank to different fixes. \*

```
int max = numbers[0];
for (int n : numbers) {
    if (n > max) {
        max = n;
    }
}
```

Mark only one oval per row.

	1 (best)	2	3 (worst)
<b>Preference</b>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
<b>Easy to read and understand</b>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
<b>Confidence in correctness</b>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
<b>Ease of modification</b>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

4. Repair option - Please rank this fix by the four criteria. Please DO NOT assign the same rank to different fixes. \*

```
int max = Integer.MIN_VALUE;  
for (int n : numbers) {  
    if (n > max) {  
        max = n;  
    }  
}
```

Mark only one oval per row.

	1 (best)	2	3 (worst)
<b>Preference</b>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
<b>Easy to read and understand</b>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
<b>Confidence in correctness</b>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
<b>Ease of modification</b>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

5. Repair option - Please rank this fix by the four criteria. Please DO NOT assign the same rank to different fixes. \*

```
int max = Arrays.stream(numbers).max().getAsInt();
```

Mark only one oval per row.

	1 (best)	2	3 (worst)
<b>Preference</b>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
<b>Easy to read and understand</b>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
<b>Confidence in correctness</b>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
<b>Ease of modification</b>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

## Task 2 Part 1

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

public class Cleaner {

    public static List<String> removeBlanks(List<String> items) {
        List<String> cleaned = items;

        for (String s : cleaned) {
            if (s.trim().isEmpty()) {
                cleaned.remove(s);
            }
        }

        return cleaned;
    }

    public static void main(String[] args) {
        List<String> list = new ArrayList<>();
        list.add("a");
        list.add(" ");
        list.add("b");

        System.out.println(removeBlanks(list));
    }
}
```

6. Please explain what the bug is with 1-2 sentences. \*

---

---

---

---

---

**Task 2 Part 2 — Please inspect all the repair options before making your decision!**

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

public class Cleaner {

    public static List<String> removeBlanks(List<String> items) {
        List<String> cleaned = items;

        for (String s : cleaned) {
            if (s.trim().isEmpty()) {
                cleaned.remove(s);
            }
        }

        return cleaned;
    }

    public static void main(String[] args) {
        List<String> list = new ArrayList<>();
        list.add("a");
        list.add(" ");
        list.add("b");

        System.out.println(removeBlanks(list));
    }
}
```

7. Repair option - Please rank this fix by the four criteria. Please DO NOT assign the same rank to different fixes. \*

```
var it = cleaned.iterator();
while (it.hasNext()) {
    String s = it.next();
    if (s.trim().isEmpty()) {
        it.remove();
    }
}
```

*Mark only one oval per row.*

	1 (best)	2	3 (worst)
<b>Preference</b>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
<b>Easy to read and understand</b>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
<b>Confidence in correctness</b>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
<b>Ease of modification</b>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

8. Repair option - Please rank this fix by the four criteria. Please DO NOT assign the same rank to different fixes. \*

```
List<String> newList = new ArrayList<>();
for (int i = 0; i < items.size(); i++) {
    String s = items.get(i);
    String trimmed = s.trim();
    boolean blank = trimmed.isEmpty();
    if (!blank) {
        newList.add(s);
    }
}
cleaned = newList;
```

Mark only one oval per row.

	1 (best)	2	3 (worst)
<b>Preference</b>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
<b>Easy to read and understand</b>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
<b>Confidence in correctness</b>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
<b>Ease of modification</b>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

9. Repair option - Please rank this fix by the four criteria. Please DO NOT assign the same rank to different fixes. \*

```
cleaned = items.stream()  
    .filter(s -> !s.trim().isEmpty())  
    .toList();
```

Mark only one oval per row.

	1 (best)	2	3 (worst)
<b>Preference</b>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
<b>Easy to read and understand</b>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
<b>Confidence in correctness</b>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
<b>Ease of modification</b>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

### Task 3 Part 1

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

public class Normalizer {

    public static List<String> normalize(List<String> items) {
        List<String> result = new ArrayList<>();

        for (int i = 0; i < items.size(); i++) {
            String s = items.get(i);

            if (s == null && s.trim().isEmpty()) {
                continue;
            }

            result.add(s.trim());
        }

        return result;
    }

    public static void main(String[] args) {
        List<String> input = List.of(" a ", null, " b");
        System.out.println(normalize(input));
    }
}
```

10. Please explain what the bug is with 1-2 sentences. \*

---

---

---

---

---

**Task 3 Part 2 — Please inspect all the repair options before making your decision!**

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

public class Normalizer {

    public static List<String> normalize(List<String> items) {
        List<String> result = new ArrayList<>();

        for (int i = 0; i < items.size(); i++) {
            String s = items.get(i);

            if (s == null && s.trim().isEmpty()) {
                continue;
            }

            result.add(s.trim());
        }

        return result;
    }

    public static void main(String[] args) {
        List<String> input = List.of(" a ", null, " b");
        System.out.println(normalize(input));
    }
}
```

11. Repair option - Please rank this fix by the four criteria. Please DO NOT assign the \* same rank to different fixes.

```
if (s == null || s.trim().isEmpty()) {  
    continue;  
}
```

*Mark only one oval per row.*

	1 (best)	2	3 (worst)
<b>Preference</b>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
<b>Easy to read and understand</b>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
<b>Confidence in correctness</b>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
<b>Ease of modification</b>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

12. Repair option - Please rank this fix by the four criteria. Please DO NOT assign the \* same rank to different fixes.

```
boolean isNull = (s == null);
boolean isEmptyAfterTrim = false;

if (!isNull) {
    String trimmed = s.trim();
    isEmptyAfterTrim = trimmed.isEmpty();
}

if (isNull || isEmptyAfterTrim) {
    continue;
}

result.add(s.trim());
```

Mark only one oval per row.

	1 (best)	2	3 (worst)
<b>Preference</b>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
<b>Easy to read and understand</b>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
<b>Confidence in correctness</b>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
<b>Ease of modification</b>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

13. Repair option - Please rank this fix by the four criteria. Please DO NOT assign the \* same rank to different fixes.

```
if (s == null) continue;  
String t = s.trim();  
if (t.isEmpty()) continue;  
result.add(t);
```

*Mark only one oval per row.*

	1 (best)	2	3 (worst)
<b>Preference</b>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
<b>Easy to read and understand</b>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
<b>Confidence in correctness</b>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
<b>Ease of modification</b>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

## Task 4 Part 1

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

public class Deduplicator {

    public static List<String> dedupe(List<String> items) {
        List<String> result = new ArrayList<>();

        for (int i = 0; i < items.size(); i++) {
            String current = items.get(i);

            boolean seen = false;
            for (int j = 0; j < result.size(); j++) {
                if (result.get(j).equals(current)) {
                    seen = true;
                }
            }

            if (!seen) {
                result.add(current);
            }
        }

        return result;
    }

    public static void main(String[] args) {
        List<String> data = List.of("a", "b", "a", "b");
        System.out.println(dedupe(data));
    }
}
```

14. Please explain what the bug is in 1-2 sentences. \*

---

---

---

---

---

**Task 4 Part 2 — Please inspect all the repair options before making your decision!**

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

public class Deduplicator {

    public static List<String> dedupe(List<String> items) {
        List<String> result = new ArrayList<>();

        for (int i = 0; i < items.size(); i++) {
            String current = items.get(i);

            boolean seen = false;
            for (int j = 0; j < result.size(); j++) {
                if (result.get(j).equals(current)) {
                    seen = true;
                }
            }

            if (!seen) {
                result.add(current);
            }
        }

        return result;
    }

    public static void main(String[] args) {
        List<String> data = List.of("a", "b", "a", "b");
        System.out.println(dedupe(data));
    }
}
```

15. Repair option - Please rank this fix by the four criteria. Please DO NOT assign the \* same rank to different fixes.

```
if (result.get(j) == null && current == null) {  
    seen = true;  
} else if (result.get(j) != null && result.get(j).equals(current)) {  
    seen = true;  
}
```

Mark only one oval per row.

	1 (best)	2	3 (worst)
<b>Preference</b>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
<b>Easy to read and understand</b>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
<b>Confidence in correctness</b>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
<b>Ease of modification</b>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

16. Repair option - Please rank this fix by the four criteria. Please DO NOT assign the \* same rank to different fixes.

```
for (int j = 0; j < result.size(); j++) {  
    String existing = result.get(j);  
  
    boolean bothNull = existing == null && current == null;  
    boolean equalNonNull = existing != null && existing.equals(current);  
  
    if (bothNull || equalNonNull) {  
        seen = true;  
        break;  
    }  
}
```

*Mark only one oval per row.*

	1 (best)	2	3 (worst)
<b>Preference</b>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
<b>Easy to read and understand</b>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
<b>Confidence in correctness</b>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
<b>Ease of modification</b>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

17. Repair option - Please rank this fix by the four criteria. Please DO NOT assign the \* same rank to different fixes.

```
for (int j = 0; j < result.size() && !seen; j++) {
    String e = result.get(j);
    if (e == current || (e != null && e.equals(current)))
        seen = true;
}
```

*Mark only one oval per row.*

Preference	1 (best)	2	3 (worst)
<b>Easy to read and understand</b>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
<b>Confidence in correctness</b>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
<b>Ease of modification</b>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

## Task 5 Part 1

The program is intended to count the number of non-empty file names in a list of files.

```
public class FileCounter {

    public static int countNonEmptyFileNames(String[] files) {
        int count = 0;
        for (String file : files) {
            count += file != null ? (file.isEmpty() ? 0 : 1) : 0;
        }
        return count;
    }

    public static void main(String[] args) {
        String[] listA = { "a.txt", "", null, "b.txt" };
        System.out.println(countNonEmptyFiles(listA));
    }
}
```

18. Please explain what the bug is in 1-2 sentences. \*

---

---

---

---

**Task 5 Part 2 — Please inspect all the repair options before making your decision!**

The program is intended to count the number of non-empty file names in a list of files.

```
public class FileCounter {

    public static int countNonEmptyFileNames(String[] files) {
        int count = 0;
        for (String file : files) {
            count += file != null ? (file.isEmpty() ? 0 : 1) : 0;
        }
        return count;
    }

    public static void main(String[] args) {
        String[] listA = { "a.txt", "", null, "b.txt" };
        System.out.println(countNonEmptyFiles(listA));
    }
}
```

19. Repair option - Please rank this fix by the four criteria. Please DO NOT assign the \* same rank to different fixes.

```
if (files == null) return 0;
```

Mark only one oval per row.

	1 (best)	2	3 (worst)
<b>Preference</b>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
<b>Easy to read and understand</b>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
<b>Confidence in correctness</b>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
<b>Ease of modification</b>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

20. Repair option - Please rank this fix by the four criteria. Please DO NOT assign the \* same rank to different fixes.

```
if (files == null) return 0;  
int count = 0;  
for (String file : files) {  
    if (file != null) {  
        if (!file.isEmpty()) {  
            count++;  
        }  
    }  
}
```

*Mark only one oval per row.*

	1 (best)	2	3 (worst)
<b>Preference</b>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
<b>Easy to read and understand</b>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
<b>Confidence in correctness</b>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
<b>Ease of modification</b>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

21. Repair option - Please rank this fix by the four criteria. Please DO NOT assign the \* same rank to different fixes.

```
if (files == null) return 0;
int count = 0;
for (String file : files) {
    if (!(file == null || file.isEmpty())) count++;
}
```

*Mark only one oval per row.*

Preference	1 (best)	2	3 (worst)
<b>Easy to read and understand</b>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
<b>Confidence in correctness</b>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
<b>Ease of modification</b>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

## Task 6 Part 1

// Pass if score is high or extra credit is granted, provided attendance is sufficient.

```
public class GradingSystem {
    public static boolean isPassing(int score, boolean extraCredit, boolean attendance) {
        return score > 70 || extraCredit && attendance;
    }

    public static void main(String[] args) {
        System.out.println(isPassing(65, true, true));
        System.out.println(isPassing(65, false, true));
        System.out.println(isPassing(75, false, false));
    }
}
```

22. Please explain what the bug is in 1-2 sentences. \*

---

---

---

---

**Task 6 Part 2 — Please inspect all the repair options before making your decision!**

// Pass if score is high or extra credit is granted, provided attendance is sufficient.

```
public class GradingSystem {  
    public static boolean isPassing(int score, boolean extraCredit, boolean attendance) {  
        return score > 70 || extraCredit && attendance;  
    }  
  
    public static void main(String[] args) {  
        System.out.println(isPassing(65, true, true));  
        System.out.println(isPassing(65, false, true));  
        System.out.println(isPassing(75, false, false));  
    }  
}
```

23. Repair option - Please rank this fix by the four criteria. Please DO NOT assign the \* same rank to different fixes.

```
return attendance && score > 70 || extraCredit;
```

*Mark only one oval per row.*

	1 (best)	2	3 (worst)
<b>Preference</b>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
<b>Easy to read and understand</b>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
<b>Confidence in correctness</b>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
<b>Ease of modification</b>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

24. Repair option - Please rank this fix by the four criteria. Please DO NOT assign the \* same rank to different fixes.

```
return (score > 70 || extraCredit) && attendance;
```

*Mark only one oval per row.*

	1 (best)	2	3 (worst)
<b>Preference</b>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
<b>Easy to read and understand</b>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
<b>Confidence in correctness</b>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
<b>Ease of modification</b>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

25. Repair option - Please rank this fix by the four criteria. Please DO NOT assign the \* same rank to different fixes.

```
if (attendance) if (score > 70 || extraCredit) return true; return false;
```

Mark only one oval per row.

	1 (best)	2	3 (worst)
<b>Preference</b>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
<b>Easy to read and understand</b>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
<b>Confidence in correctness</b>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
<b>Ease of modification</b>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

## Task 7 Part 1

```
public class Counter {  
    private int total = 0;  
  
    public void addValues() {  
        int[] values = {2, 4, 6};  
        int i = 0;  
  
        while (i < values.length) {  
            total += ++i * 2;  
            if (total % 3 == 0)  
                total += 1;  
        }  
    }  
  
    public void printTotal() {  
        System.out.println("Total: " + total);  
    }  
  
    public static void main(String[] args) {  
        Counter counter = new Counter();  
        counter.addValues();  
        counter.printTotal();  
    }  
}
```

26. Please explain what the bug is in 1-2 sentences. \*

---

---

---

---

---

**Task 7 Part 2 — Please inspect all the repair options before making your decision!**

```
public class Counter {  
    private int total = 0;  
  
    public void addValues() {  
        int[] values = {2, 4, 6};  
        int i = 0;  
  
        while (i < values.length) {  
            total += ++i * 2;  
            if (total % 3 == 0)  
                total += 1;  
            }  
    }  
  
    public void printTotal() {  
        System.out.println("Total: " + total);  
    }  
  
    public static void main(String[] args) {  
        Counter counter = new Counter();  
        counter.addValues();  
        counter.printTotal();  
    }  
}
```

27. Repair option - Please rank this fix by the four criteria. Please DO NOT assign the \* same rank to different fixes.

```
while (i < values.length) {  
    int temp = ++i;  
    total += temp * 2;  
    if (total % 3 == 0)  
        total += 1;  
}
```

Mark only one oval per row.

	1 (best)	2	3 (worst)
<b>Preference</b>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
<b>Easy to read and understand</b>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
<b>Confidence in correctness</b>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
<b>Ease of modification</b>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

28. Repair option - Please rank this fix by the four criteria. Please DO NOT assign the \* same rank to different fixes.

```
while (i < values.length) {  
    i++;  
    total += i * 2;  
    if (total % 3 == 0)  
        total += 1;  
}
```

Mark only one oval per row.

	1 (best)	2	3 (worst)
<b>Preference</b>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
<b>Easy to read and understand</b>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
<b>Confidence in correctness</b>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
<b>Ease of modification</b>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

29. Repair option - Please rank this fix by the four criteria. Please DO NOT assign the \* same rank to different fixes.

```
for (int j = 1; j <= values.length; j++) {
    total += j * 2 + ((total + j * 2) % 3 == 0 ? 1 : 0);
}
```

*Mark only one oval per row.*

Preference	1 (best)	2	3 (worst)
<b>Easy to read and understand</b>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
<b>Confidence in correctness</b>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
<b>Ease of modification</b>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

## Task 8 Part 1

```
public class Converter {
    public void convertValues() {
        int[] numbers = {100, 200, 300};
        byte[] results = new byte[numbers.length];

        for (int i = 0; i < numbers.length; i++) {
            results[i] = (byte) numbers[i];
        }

        for (int i = 0; i < results.length; i++) {
            System.out.println(results[i]);
        }
    }

    public static void main(String[] args) {
        Converter c = new Converter();
        c.convertValues();
    }
}
```

30. Please explain what the bug is in 1-2 sentences. \*

---

---

---

---

**Task 8 Part 2 — Please inspect all the repair options before making your decision!**

```
public class Converter {  
    public void convertValues() {  
        int[] numbers = {100, 200, 300};  
        byte[] results = new byte[numbers.length];  
  
        for (int i = 0; i < numbers.length; i++) {  
            results[i] = (byte) numbers[i];  
        }  
  
        for (int i = 0; i < results.length; i++) {  
            System.out.println(results[i]);  
        }  
    }  
  
    public static void main(String[] args) {  
        Converter c = new Converter();  
        c.convertValues();  
    }  
}
```

31. Repair option - Please rank this fix by the four criteria. Please DO NOT assign the \* same rank to different fixes.

```
for (int i = 0; i < numbers.length; i++) {  
    int temp = numbers[i];  
    results[i] = (byte) temp;  
}
```

*Mark only one oval per row.*

	1 (best)	2	3 (worst)
<b>Preference</b>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
<b>Easy to read and understand</b>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
<b>Confidence in correctness</b>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
<b>Ease of modification</b>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

32. Repair option - Please rank this fix by the four criteria. Please DO NOT assign the \* same rank to different fixes.

```
for (int i = 0; i < numbers.length; i++) {  
    results[i] = numbers[i] > Byte.MAX_VALUE ? Byte.MAX_VALUE : (numbers[i] <  
Byte.MIN_VALUE ? Byte.MIN_VALUE : (byte) numbers[i]);  
}
```

*Mark only one oval per row.*

	1 (best)	2	3 (worst)
<b>Preference</b>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
<b>Easy to read and understand</b>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
<b>Confidence in correctness</b>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
<b>Ease of modification</b>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

33. Repair option - Please rank this fix by the four criteria. Please DO NOT assign the \* same rank to different fixes.

```
for (int i = 0; i < numbers.length; i++) {  
    results[i] = (numbers[i] % 128 < 0 ? (byte)(numbers[i] + 128) : (byte)numbers[i]);  
}
```

Mark only one oval per row.

	1 (best)	2	3 (worst)
<b>Preference</b>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
<b>Easy to read and understand</b>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
<b>Confidence in correctness</b>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
<b>Ease of modification</b>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

---

This content is neither created nor endorsed by Google.

Google Forms

