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: **30–45 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**.

Check all that apply.

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

Task 1 Part 1

This Java method finds the largest number in an array.

```
public class BaselineExample {
    public static int findMax(int[] numbers) {
        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) {
      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 t different fixes.

   ```
   int max = numbers[0];
   ```

   Mark only one oval per row.

	1 (best)	2	3 (worst)
Preference	⬭	⬭	⬭
Easy to read and understand	⬭	⬭	⬭
Confidence in correctness	⬭	⬭	⬭
Ease of modification	⬭	⬭	⬭

4. Repair option - Please rank this fix by the four criteria. Please DO NOT assign the same rank t 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)
Preference	◯	◯	◯
Easy to read and understand	◯	◯	◯
Confidence in correctness	◯	◯	◯
Ease of modification	◯	◯	◯

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

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

Mark only one oval per row.

	1 (best)	2	3 (worst)
Preference	◯	◯	◯
Easy to read and understand	◯	◯	◯
Confidence in correctness	◯	◯	◯
Ease of modification	◯	◯	◯

Task 2 Part 1

The file is readable if the 3rd decimal places is 1.

```
public class PermissionChecker {
    public static String hasReadPermission(int permissions) {
        int readable = 4;
        if (permissions & readable != 0) {
            return "readable";
        } else {
            return "unreadable, permission denied";
        }
    }

    public static void main(String[] args) {
        int permissions = 777 & 555;
        System.out.println("Read permission: " + hasReadPermission(permissions));
    }
}
```

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

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

The file is readable if the 3rd decimal places is 1.

```
public class PermissionChecker {
   public static String hasReadPermission(int permissions) {
      int readable = 4;
      if (permissions & readable != 0) {
         return "readable";
      } else {
         return "unreadable, permission denied";
      }
   }

   public static void main(String[] args) {
      int permissions = 777 & 555;
      System.out.println("Read permission: " + hasReadPermission(permissions));
   }
}
```

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

```
if (permissions & readable == readable) {
   return "readable";
} else {
   return "unreadable, permission denied";
}
```

Mark only one oval per row.

	1 (best)	2	3 (worst)
Preference	◯	◯	◯
Easy to read and understand	◯	◯	◯
Confidence in correctness	◯	◯	◯
Ease of modification	◯	◯	◯

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

```
int readable = 0b100;
if (permissions & readable == readable) {
   return "readable";
} else {
   return "unreadable, permission denied";
}
```

Mark only one oval per row.

	1 (best)	2	3 (worst)
Preference	◯	◯	◯
Easy to read and understand	◯	◯	◯
Confidence in correctness	◯	◯	◯
Ease of modification	◯	◯	◯

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

```
String[ ] readability = {"unreadable, permission denied", "readable"};
return readability[0b1 & permissions >> 2];
```

Mark only one oval per row.

	1 (best)	2	3 (worst)
Preference	◯	◯	◯
Easy to read and understand	◯	◯	◯
Confidence in correctness	◯	◯	◯
Ease of modification	◯	◯	◯

Task 3 Part 1

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

```
public class FileCounter {

   public static int countNonEmptyFiles(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));
   }
}
```

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

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

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

```
public class FileCounter {

    public static int countNonEmptyFiles(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)); // 2
    }
}
```

11. 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)
Preference	◯	◯	◯
Easy to read and understand	◯	◯	◯
Confidence in correctness	◯	◯	◯
Ease of modification	◯	◯	◯

12. 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)
Preference	⬭	⬭	⬭
Easy to read and understand	⬭	⬭	⬭
Confidence in correctness	⬭	⬭	⬭
Ease of modification	⬭	⬭	⬭

13. 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.

	1 (best)	2	3 (worst)
Preference	⬭	⬭	⬭
Easy to read and understand	⬭	⬭	⬭
Confidence in correctness	⬭	⬭	⬭
Ease of modification	⬭	⬭	⬭

Task 4 Part 1

This function validates whether the password matches what is stored.

```
public class PasswordValidator {
   public static boolean validate(String password) {
      if (password.length() > 8)
         if (password.matches(".*\\d.*"))
            System.out.println("Valid password");
            return true;
      return false;
   }

   public static void main(String[] args) {
      System.out.println(validate("abc123"));
   }
}
```

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!

This function validates whether the password matches what is stored.

```
public class PasswordValidator {
   public static boolean validate(String password) {
      if (password.length() > 8)
         if (password.matches(".*\\d.*"))
            System.out.println("Valid password");
            return true;
      return false;
   }

   public static void main(String[] args) {
      System.out.println(validate("abc123"));
   }
}
```

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

```
if (password.length() > 8)
   if (password.matches(".*\\d.*"))
      return true;
return false;
```

Mark only one oval per row.

	1 (best)	2	3 (worst)
Preference	◯	◯	◯
Easy to read and understand	◯	◯	◯
Confidence in correctness	◯	◯	◯
Ease of modification	◯	◯	◯

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

```
if (password.length() > 8)
   if (password.matches(".*\\d.*")) {
      System.out.println("Valid password");
      return true;
   }
return false;
```

Mark only one oval per row.

	1 (best)	2	3 (worst)
Preference	◯	◯	◯
Easy to read and understand	◯	◯	◯
Confidence in correctness	◯	◯	◯
Ease of modification	◯	◯	◯

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

return password.length() > 8 ? (password.matches(".*\\d.*") ? true : false) : false;

Mark only one oval per row.

	1 (best)	2	3 (worst)
Preference	◯	◯	◯
Easy to read and understand	◯	◯	◯
Confidence in correctness	◯	◯	◯
Ease of modification	◯	◯	◯

Task 5 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));
   }
}
```

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!

```
// 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));
    }
}
```

19. 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)
Preference	◯	◯	◯
Easy to read and understand	◯	◯	◯
Confidence in correctness	◯	◯	◯
Ease of modification	◯	◯	◯

20. 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)
Preference	◯	◯	◯
Easy to read and understand	◯	◯	◯
Confidence in correctness	◯	◯	◯
Ease of modification	◯	◯	◯

21. 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)
Preference	◯	◯	◯
Easy to read and understand	◯	◯	◯
Confidence in correctness	◯	◯	◯
Ease of modification	◯	◯	◯

Task 6 Part 1

```
public class UserManager {
    private boolean[] activeFlags;
    private int userCount;

    public UserManager(int size) {
        activeFlags = new boolean[size];
        userCount = 0;
    }

    private boolean hasActiveUsers() {
        for (boolean flag : activeFlags) {
            if (flag) return true;
        }
        return false;
    }

    private boolean activateNextUser() {
        for (int i = 0; i < activeFlags.length; i++) {
            if (!activeFlags[i]) {
                activeFlags[i] = true;
                userCount++;
                return true;
            }
        }
        return false;
    }

    public void ensureUserActive() {
        if (hasActiveUsers() || activateNextUser()) {
            System.out.println("User active. Total active: " + userCount);
        }
    }

    public static void main(String[] args) {
        UserManager manager = new UserManager(1);
        manager.ensureUserActive();
        manager.ensureUserActive();
    }
}
```

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!

```
public class UserManager {
   private boolean[] activeFlags;
   private int userCount;

   public UserManager(int size) {
      activeFlags = new boolean[size];
      userCount = 0;
   }

   private boolean hasActiveUsers() {
      for (boolean flag : activeFlags) {
         if (flag) return true;
      }
      return false;
   }

   private boolean activateNextUser() {
      for (int i = 0; i < activeFlags.length; i++) {
         if (!activeFlags[i]) {
            activeFlags[i] = true;
            userCount++;
            return true;
         }
      }
      return false;
   }

   public void ensureUserActive() {
      if (hasActiveUsers() || activateNextUser()) {
         System.out.println("User active. Total active: " + userCount);
      }
   }

   public static void main(String[] args) {
      UserManager manager = new UserManager(1);
      manager.ensureUserActive();
      manager.ensureUserActive();
   }
}
```

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

```
public void ensureUserActive() {
   boolean unused = hasActiveUsers() || alwaysActivate();
   System.out.println("User active. Total active: " + userCount);
}

private boolean alwaysActivate() {
   activateNextUser();
   return true;
}
```

Mark only one oval per row.

	1 (best)	2	3 (worst)
Preference	◯	◯	◯
Easy to read and understand	◯	◯	◯
Confidence in correctness	◯	◯	◯
Ease of modification	◯	◯	◯

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

```
public void ensureUserActive() {
   if (!hasActiveUsers()) {
      activateNextUser();
   }
   System.out.println("User active. Total active: " + userCount);
}
```

Mark only one oval per row.

	1 (best)	2	3 (worst)
Preference	◯	◯	◯
Easy to read and understand	◯	◯	◯
Confidence in correctness	◯	◯	◯
Ease of modification	◯	◯	◯

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

```
public void ensureUserActive() {
   boolean _ = hasActiveUsers() ? true : activateNextUser();
   System.out.println("User active. Total active: " + userCount);
}
```

Mark only one oval per row.

	1 (best)	2	3 (worst)
Preference	◯	◯	◯
Easy to read and understand	◯	◯	◯
Confidence in correctness	◯	◯	◯
Ease of modification	◯	◯	◯

Task 7 Part 1

The method is intended to send notifications to active users only.

```
public class NotificationSender {
    public static void sendNotification(boolean isActive, boolean isPremium) {
        if (isActive)
            if (isPremium)
                System.out.println("Premium user notified");
            else
                System.out.println("Regular user notified");
        System.out.println("Notification process complete");
    }

    public static void main(String[] args) {
        sendNotification(true, false);
    }
}
```

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 OrderProcessor {
   private int processed = 0;
   private int failed = 0;

   public void processOrder(boolean priority, boolean special) {
      if (priority)
         processed += 10;
         failed += 1;

      if (special)
         if (processed > 0)
            processed += 5;
         failed += 2;

      System.out.println("Processed: " + processed + ", Failed: " + failed);
   }

   public static void main(String[] args) {
      OrderProcessor op = new OrderProcessor();
      op.processOrder(false, true);
   }
}
```

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

```
public void processOrder(boolean priority, boolean special) {
   if (priority)
      processed += 10;
   if (priority)
      failed += 0; // clarify intent, still OCB style

   if (special)
      if (processed > 0)
         processed += 5;
   if (special && processed == 0)
      failed += 2; // ensures logic matches intent

   System.out.println("Processed: " + processed + ", Failed: " + failed);
}
```

Mark only one oval per row.

	1 (best)	2	3 (worst)
Preference	◯	◯	◯
Easy to read and understand	◯	◯	◯
Confidence in correctness	◯	◯	◯
Ease of modification	◯	◯	◯

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

```
public void processOrder(boolean priority, boolean special) {
    if (priority) {
        processed += 10;
        failed += 1;
    }

    if (special) {
        if (processed > 0) {
            processed += 5;
        } else {
            failed += 2;
        }
    }

    System.out.println("Processed: " + processed + ", Failed: " + failed);
}
```

Mark only one oval per row.

	1 (best)	2	3 (worst)
Preference	◯	◯	◯
Easy to read and understand	◯	◯	◯
Confidence in correctness	◯	◯	◯
Ease of modification	◯	◯	◯

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

```
public void processOrder(boolean priority, boolean special) {
   processed += priority ? 10 : 0;
   failed += priority ? 1 : 0;

   processed += (special && processed > 0) ? 5 : 0;
   failed += (special && processed == 0) ? 2 : 0;

   System.out.println("Processed: " + processed + ", Failed: " + failed);
}
```

Mark only one oval per row.

	1 (best)	2	3 (worst)
Preference			
Easy to read and understand			
Confidence in correctness			
Ease of modification			

Task 8 Part 1

```
public class ScoreTracker {
    private int total = 0;

    public void addScores() {
        int[] points = {5, 10, 15};
        int i = 0;

        while (i < points.length) {
            total += points[i++] * 2;
            if (total > 20)
                total += 1;
        }
    }

    public void printTotal() {
        System.out.println("Total: " + total);
    }

    public static void main(String[] args) {
        ScoreTracker st = new ScoreTracker();
        st.addScores();
        st.printTotal();
    }
}
```

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 ScoreTracker {
    private int total = 0;

    public void addScores() {
        int[] points = {5, 10, 15};
        int i = 0;

        while (i < points.length) {
            total += points[i++] * 2;
            if (total > 20)
                total += 1;
        }
    }

    public void printTotal() {
        System.out.println("Total: " + total);
    }

    public static void main(String[] args) {
        ScoreTracker st = new ScoreTracker();
        st.addScores();
        st.printTotal();
    }
}
```

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

```
while (i < points.length) {
   int temp = points[i++];
   total += temp * 2;
   if (total > 20)
      total += 1;
}
```

Mark only one oval per row.

	1 (best)	2	3 (worst)
Preference	◯	◯	◯
Easy to read and understand	◯	◯	◯
Confidence in correctness	◯	◯	◯
Ease of modification	◯	◯	◯

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

```
while (i < points.length) {
   total += points[i] * 2;
   i++;
   if (total > 20)
      total += 1;
}
```

Mark only one oval per row.

	1 (best)	2	3 (worst)
Preference	⬭	⬭	⬭
Easy to read and understand	⬭	⬭	⬭
Confidence in correctness	⬭	⬭	⬭
Ease of modification	⬭	⬭	⬭

33. 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 < points.length; j++) {
   total += points[j] * 2 + (total + points[j] * 2 > 20 ? 1 : 0);
}
```

Mark only one oval per row.

	1 (best)	2	3 (worst)
Preference	⬭	⬭	⬭
Easy to read and understand	⬭	⬭	⬭
Confidence in correctness	⬭	⬭	⬭
Ease of modification	⬭	⬭	⬭

Task 9 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();
    }
}
```

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

Task 9 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();
    }
}
```

35. 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)
Preference	⬭	⬭	⬭
Easy to read and understand	⬭	⬭	⬭
Confidence in correctness	⬭	⬭	⬭
Ease of modification	⬭	⬭	⬭

36. 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)
Preference	⬭	⬭	⬭
Easy to read and understand	⬭	⬭	⬭
Confidence in correctness	⬭	⬭	⬭
Ease of modification	⬭	⬭	⬭

37. 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.

	1 (best)	2	3 (worst)
Preference	⬭	⬭	⬭
Easy to read and understand	⬭	⬭	⬭
Confidence in correctness	⬭	⬭	⬭
Ease of modification	⬭	⬭	⬭

Task 10 Part 1

```
public class ArrayProcessor {
   public void process() {
      int[] data = {1, 2, 3, 4};
      int temp = 0;

      for (int i = 0; i < data.length; i++) {
         temp = i;
         for (int j = 0; j < data.length; j++) {
            data[j] += temp;
         }
      }

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

   public static void main(String[] args) {
      ArrayProcessor ap = new ArrayProcessor();
      ap.process();
   }
}
```

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

Task 10 Part 2 — Please inspect all the repair options before making your decision!

The method fills a stack in a certain pattern.

```
public class StackSimulator {

    public static void fillStack(int[] stack) {
        stack[4] = 5;

        while (stack[4] > 0) {
            stack[stack[4] - 1] = stack[4];
            stack[4] = stack[4] - 1;
        }
    }

    public static void main(String[] args) {
        int[] stack = new int[5];
        fillStack(stack);
        for (int i = 0; i < stack.length; i++) {
            System.out.print(stack[i] + " ");
        }
    }
}
```

39. 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 < data.length; i++) {
   temp = i;
   for (int j = 0; j < data.length; j++) {
      int temp2 = temp;
      data[j] += temp2;
   }
}
```

Mark only one oval per row.

	1 (best)	2	3 (worst)
Preference	⬭	⬭	⬭
Easy to read and understand	⬭	⬭	⬭
Confidence in correctness	⬭	⬭	⬭
Ease of modification	⬭	⬭	⬭

40. 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 < data.length; i++) {
   int current = i;
   for (int j = 0; j < data.length; j++) {
      data[j] += current;
   }
}
```

Mark only one oval per row.

	1 (best)	2	3 (worst)
Preference	⬭	⬭	⬭
Easy to read and understand	⬭	⬭	⬭
Confidence in correctness	⬭	⬭	⬭
Ease of modification	⬭	⬭	⬭

41. 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 < data.length; i++) {
   for (int j = 0; j < data.length; j++) {
      data[j] += i % 2 == 0 ? i : 0;
   }
}
```

Mark only one oval per row.

	1 (best)	2	3 (worst)
Preference	◯	◯	◯
Easy to read and understand	◯	◯	◯
Confidence in correctness	◯	◯	◯
Ease of modification	◯	◯	◯

Task 11 Part 1

The method calculates the total price of items.

```
public class PriceCalculator {

   public static int totalPrice(double[] prices) {
      int total = 0;
      for (int i = 0; i < prices.length; i++) {
         total += (int) prices[i];
      }
      return total;
   }

   public static void main(String[] args) {
      double[] prices = {1.5, 2.3, 3.7};
      System.out.println(totalPrice(prices));
   }
}
```

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

Task 11 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();
   }
}
```

43. 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)
Preference	◯	◯	◯
Easy to read and understand	◯	◯	◯
Confidence in correctness	◯	◯	◯
Ease of modification	◯	◯	◯

44. 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)
Preference	◯	◯	◯
Easy to read and understand	◯	◯	◯
Confidence in correctness	◯	◯	◯
Ease of modification	◯	◯	◯

45. 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)
Preference	◯	◯	◯
Easy to read and understand	◯	◯	◯
Confidence in correctness	◯	◯	◯
Ease of modification	◯	◯	◯