Java Control Flow and Algorithms - Summary

Control Flow Statements in Java

Java uses control flow statements to break the normal top-to-bottom execution flow, allowing conditional code execution, loops, and branching. The key control flow structures in Java are:

1. Decision-Making Statements: if, if-else, switch

2. Looping Statements: for, while, do-while

3. Branching Statements: break, continue, return

1. If and If-Else Statements:

- The if statement allows you to execute a block of code if a condition is true.

- The if-else statement gives an alternate path if the condition is false.

Example:

```
if (isMoving) {
    currentSpeed--;
} else {
    System.out.println("The bicycle has stopped.");
}
```

2. Switch Statement:

- Switch statements can have multiple cases for a single variable.

- It works with int, char, byte, short, enums, and Strings.

- A break is essential to prevent fall-through between cases.

Example:

```
switch (month) {
    case 1: System.out.println("January");
            break;
    case 2: System.out.println("February");
```

```
        break;
    default: System.out.println("Invalid month");
}
```

3. While and Do-While Statements:

- while runs a loop as long as the condition is true.

- do-while ensures the code runs at least once, then checks the condition.

Example (while):

```
int count = 1;
while (count <= 10) {
    System.out.println(count);
    count++;
}
```

Example (do-while):

```
int count = 1;
do {
    System.out.println(count);
    count++;
} while (count <= 10);
```

4. For Statement:

- for loops are compact and iterate over ranges.

- Enhanced for loops are useful for iterating through arrays.

Example (for):

```
for (int i = 0; i < 10; i++) {
    System.out.println(i);
}
```

Example (Enhanced for):

```
int[] numbers = {1, 2, 3, 4, 5};

for (int num : numbers) {

    System.out.println(num);

}
```

5. Branching Statements:

- break exits a loop or switch early.

- continue skips the current iteration of a loop.

- return exits a method.

Example (break):

```
for (int i = 0; i < 10; i++) {

    if (i == 5) break;

    System.out.println(i);

}
```

Algorithms:

1. Sequential Search:

Iterates through an array looking for a value. Simple but inefficient for large datasets.

Example:

```
for (int i = 0; i < array.length; i++) {

    if (array[i] == target) {

        System.out.println("Found at index: " + i);

        break;

    }

}
```

2. Binary Search:

Efficient search in sorted arrays. Divides the array and checks the middle value.

Example:

```
int lower = 0;
int upper = array.length - 1;
while (lower <= upper) {
    int mid = (lower + upper) / 2;
    if (array[mid] == target) {
        System.out.println("Found at index: " + mid);
        break;
    } else if (array[mid] < target) {
        lower = mid + 1;
    } else {
        upper = mid - 1;
    }
}
```

3. Bubble Sort:

Compares each element in a list and swaps them if they're out of order. It's simple but not efficient.

Example:

```
for (int i = 0; i < array.length - 1; i++) {
    for (int j = 0; j < array.length - 1 - i; j++) {
        if (array[j] > array[j+1]) {
            int temp = array[j];
            array[j] = array[j+1];
            array[j+1] = temp;
        }
    }
}
```