Minimizing a Java program typically involves optimizing the code for readability, performance, and maintainability. It can include removing redundant code, simplifying expressions, reducing memory usage, and improving efficiency. Here are some common techniques with examples:

1. **Removing Unused Imports**

Unused imports increase code clutter and memory usage. Removing them makes the program more efficient and reduces compile time.

Before:

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

public class Example {
    public static void main(String[] args) {
        List<String> list = new ArrayList<>();
        list.add("Hello");
        System.out.println(list.get(0));
    }
}
```

After:

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

public class Example {
```

```
    public static void main(String[] args) {
        List<String> list = new ArrayList<>();
        list.add("Hello");
        System.out.println(list.get(0));
    }
}
```

2. **Inlining Simple Methods**

If a method is only used in one place and is straightforward, you can inline it (write it directly) to avoid unnecessary function calls.

Before:

```java
public class Example {
    public static void main(String[] args) {
        printMessage("Hello, World!");
    }

    private static void printMessage(String message) {
        System.out.println(message);
    }
}
```

After:

```java
public class Example {
    public static void main(String[] args) {
        System.out.println("Hello, World!");
    }
}
```

3. **Using `StringBuilder` for String Concatenation**

In loops, avoid using `String` concatenation as it creates multiple objects. Instead, use `StringBuilder`.

Before:

```java
public class Example {
    public static void main(String[] args) {
        String result = "";
        for (int i = 0; i < 10; i++) {
            result += i;
        }
        System.out.println(result);
    }
}
```

After:

```java
public class Example {
    public static void main(String[] args) {
        StringBuilder result = new StringBuilder();
        for (int i = 0; i < 10; i++) {
            result.append(i);
        }
        System.out.println(result.toString());
    }
}
```

4. **Minimizing Object Creation**

Avoid creating unnecessary objects, especially in loops, as it increases memory consumption.

Before:

```java
public class Example {
    public static void main(String[] args) {
        for (int i = 0; i < 10; i++) {
            Integer num = new Integer(i);
            System.out.println(num);
        }
    }
}
```

```

**After:**

```java
public class Example {
    public static void main(String[] args) {
        for (int i = 0; i < 10; i++) {
            System.out.println(i);
        }
    }
}
```

5. **Avoiding Redundant Code**

Redundant code can make programs slower and harder to read.

Before:

```java
public class Example {
    public static void main(String[] args) {
        int x = 5;
        if (x > 0) {
            System.out.println("Positive");
        } else if (x < 0) {
            System.out.println("Negative");
```
```

```
        } else if (x == 0) {
            System.out.println("Zero");
        }
    }
}
```

After:

```java
public class Example {
    public static void main(String[] args) {
        int x = 5;
        if (x > 0) {
            System.out.println("Positive");
        } else if (x < 0) {
            System.out.println("Negative");
        } else {
            System.out.println("Zero");
        }
    }
}
```

6. **Optimizing Loops**

Using enhanced `for` loops can reduce code length and make it more readable.

Before:

```java
import java.util.ArrayList;

public class Example {
    public static void main(String[] args) {
        ArrayList<Integer> numbers = new ArrayList<>();
        for (int i = 0; i < numbers.size(); i++) {
            System.out.println(numbers.get(i));
        }
    }
}
```

After:

```java
import java.util.ArrayList;

public class Example {
    public static void main(String[] args) {
        ArrayList<Integer> numbers = new ArrayList<>();
        for (int number : numbers) {
            System.out.println(number);
        }
```

```
    }
}
```

By using these techniques, you can create a Java program that is more efficient, readable, and maintainable.