Ja, in der Methode `pressBinaryOperationKey` scheint tatsächlich etwas zu fehlen, um die im Kommentar beschriebene Funktionalität vollständig umzusetzen. Die Methode ist im Moment sehr rudimentär und legt lediglich fest, welche Operation der Benutzer gerade gedrückt hat. Die vollständige Implementierung sollte Folgendes umfassen, um der Beschreibung gerecht zu werden:

1. Speichern des vorherigen Werts: Da es sich um eine binäre Operation handelt, würde man normalerweise zwei Zahlen benötigen: die momentan eingegebene Zahl (bereits gespeichert als `latestValue`) und eine weitere Zahl (zweiter Operand), die der Nutzer nach Festlegen der Operation eingibt.

2. Speichern und Ausführen der Operation: Nach dem Drücken einer Operationstaste sollte die Methode die aktuelle Operation speichern. Nach Eintritt einer zweiten Zahl und eines nochmaligen Drückens der Operationstaste müsste die Methode die Operation ausführen und das Resultat anzeigen.

3. Berechnung implementieren: Folgender Pseudocode beschreibt den logischen Fluss für das Ausführen der ausgewählten Operation:

java

public void pressBinaryOperationKey(String operation) {

if (latestOperation != null) { // Wenn bereits eine Operation festgelegt ist

double currentValue = Double.parseDouble(screen);

double result;

switch (latestOperation) {

case "+":

result = latestValue + currentValue;

break;

case "-":

result = latestValue - currentValue;

break;

case "x":

result = latestValue \* currentValue;

break;

case "/":

if (currentValue == 0) {

screen = "Error";

latestOperation = null; // Um den Zustand nach Fehler zurückzusetzen

return;

} else {

result = latestValue / currentValue;

}

break;

default:

throw new IllegalArgumentException("Invalid operation");

}

screen = Double.toString(result);

}

// Neue Operation speichern

latestOperation = operation;

latestValue = Double.parseDouble(screen); // Das Zwischenergebnis wird neuer erster Operand

}

Wichtige Aspekte:

- Erste Operation: Beim ersten Drücken einer Operationstaste speichert die Methode einfach die derzeitige Eingabe und wartet auf die zweite Zahl.

- Zweite Eingabe und Operation: Beim zweiten Drücken nach Eingabe eines zweiten Wertes wird die gespeicherte Operation ausgeführt.

- Fehlerbehandlung: Es gibt eine spezielle Überprüfung für die Division durch Null, die den Bildschirm auf "Error" setzt, um die Anforderung aus der Kommentarzeile zu erfüllen.

- Zustandsverwaltung: Nach Abschluss einer Operation wird die aktuelle Operation zurückgesetzt, und ein neues Ergebnis kann als Ausgangspunkt für eine neue Berechnung verwendet werden.

Diese Methode stellt somit sicher, dass alle typischen Anforderungen einer einfachen Rechenkernlogik für binäre Operationen erfüllt werden.Gute Fragen! Lassen Sie uns diese nacheinander durchgehen:

1. Verwendung von `return`:

- Die `return`-Anweisung wird hier verwendet, um die Methode vorzeitig zu beenden, wenn eine Division durch Null erkannt wird. Sobald "Error" angezeigt wird, möchten wir die restlichen Verarbeitungsschritte in der Methode überspringen, da keine gültige Berechnung durchgeführt werden kann. Der `return` sorgt dafür, dass die Methode an dieser Stelle aufhört.

2. Umstellung auf den modernen `switch`-Ausdruck mit `->`-Syntax:

- Ja, es ist möglich, einen modernen `switch`-Ausdruck zu verwenden. Die `->`-Syntax ist oft kompakter und reduziert die Notwendigkeit von `break`-Anweisungen. Hier ein Beispiel:

```java

public void pressBinaryOperationKey(String operation) {

if (latestOperation != null) {

double currentValue = Double.parseDouble(screen);

double result;

result = switch (latestOperation) {

case "+" -> latestValue + currentValue;

case "-" -> latestValue - currentValue;

case "x" -> latestValue \* currentValue;

case "/" -> {

if (currentValue == 0) {

screen = "Error";

latestOperation = null;

yield Double.NaN; // Ungültiger Wert, um Fehlerzustand zu signalisieren

} else {

yield latestValue / currentValue;

}

}

default -> throw new IllegalArgumentException("Invalid operation");

};

if (!Double.isNaN(result)) { // Nur aktualisieren, wenn kein Fehler

screen = Double.toString(result);

}

}

// Neue Operation speichern

latestOperation = operation;

latestValue = Double.parseDouble(screen);

}

```

- In diesem Beispiel wird `yield` verwendet, um einen Wert aus einem blockartigen `case` zurückzugeben, insbesondere beim Teilen. Da ein Fall für eine Division durch Null berücksichtigt werden muss, gehen wir sicher, dass nur gültige Ergebnisse auf dem Bildschirm angezeigt werden.

3. Fehlen einer `else`-Anweisung:

- Eine `else`-Anweisung ist in diesem Fall nicht notwendig. Dies liegt daran, dass die Aktualisierung `latestOperation` und `latestValue` in jedem Fall passieren soll, nachdem eine Operation gedrückt wurde, unabhängig davon, ob vorher schon eine Rechnung stattfand. Das ermöglicht, dass beim Drücken der Operationstaste nicht nur gerechnet wird, sondern auch die neue Operation gespeichert wird – ohne Bedingung.

- Das Vorhandensein von `else` erübrigt sich hier, da die beiden Zeilen am Ende in jedem Ausführungspfad der Methode erreicht werden sollen. Das entspricht dem Konzept, dass die aktuelle Zahl (oder das Ergebnis einer Berechnung) zum neuen ersten Operand wird, sobald eine weitere Operation gedrückt wird.

Ich hoffe, diese Erklärungen klären Ihre Frage und helfen Ihnen zu verstehen, wie die Methode funktionieren könnte. Bei weiteren Fragen stehe ich gerne zur Verfügung!

Mindfart

double result = switch(operation) {  
 case "+" -> latestValue + Double.*parseDouble*(screen);  
 case "-" -> latestValue - Double.*parseDouble*(screen);  
 case "x" -> latestValue \* Double.*parseDouble*(screen);  
 case "/" -> {  
 if (latestValue == 0) {  
 screen = "Division durch Null nicht möglich";  
 latestOperation = null;  
 yield Double.*NaN*;  
 } else {  
 yield latestValue / Double.*parseDouble*(screen);  
 }  
 }  
 default -> throw new IllegalArgumentException();  
  
  
};  
latestValue = Double.*parseDouble*(screen);  
latestOperation = operation;