EECS 1001 and EECS 1030, lab 01 Conflict

Those students who are taking EECS 1001 and who are enrolled in lab 01 of EECS 1030 should switch to lab 02.

Lab1

The deadline for lab 1 has been extended: submit your solution **before** Saturday January 17.

When you submit you should see

```
Your code compiled successfully.
Your code passed all the tests.
Your code contains no style errors.
Your group.txt file has been successfully validated.
```

If in doubt whether you have seen that, please submit again.

Quiz

Take a piece of paper and a pen or pencil.

Question 1

Declare and initialize the field named `CENTS_PER_QUARTER`, which is a constant and has value 25.

Question 2

For the class `Converter`, complete the method

```
public static int quarterToCents(int number)
```

which given a `number` of quarters, return the corresponding number of cents.

Question 3

Why do we add a private constructor to the `Converter` class?

Question 4

Assuming that `number >= 0 && number <= 1000` is its precondition, give a test vector for the `quarterToCents` method.

Question 5

How do you document the following precondition?

```
number >= 0 && number <= 1000
```

Question 6

How many hours in total (including lecture time and lab time) did you spend on this course during the first week of the term?

Question 1

Question

Declare and initialize the field named `CENTS_PER_QUARTER`, which is a constant and has value 25.

Question 1

Question

Declare and initialize the field named `CENTS_PER_QUARTER`, which is a constant and has value 25.

Answer

```
public static final int CENTS_PER_QUARTER = 25;
```

Question 1

Question

Declare and initialize the field named `CENTS_PER_QUARTER`, which is a constant and has value 25.

Answer

```
public static final int CENTS_PER_QUARTER = 25;
```

Marking scheme

1 mark for public (fields are public attributes), 1 mark for final (constants are final), 1 mark for the rest.

Question 2

Question

For the class `Converter`, complete the method

```
public static int quarterToCents(int number)
```

which given a `number` of quarters, return the corresponding number of cents.

Question 2

Question

For the class `Converter`, complete the method

```
public static int quarterToCents(int number)
```

which given a `number` of quarters, return the corresponding number of cents.

Answer

```
return Converter.CENTS_PER_QUARTER * number;
```

Question 2

Question

For the class `Converter`, complete the method

```
public static int quarterToCents(int number)
```

which given a `number` of quarters, return the corresponding number of cents.

Answer

```
return Converter.CENTS_PER_QUARTER * number;
```

Marking scheme

1 mark for the return statement, 1 mark for using the class name, 1 mark for the rest.

Question 3

Question

Why do we add a private constructor to the `Converter` class?

Question 3

Question

Why do we add a private constructor to the `Converter` class?

Answer

Otherwise, the compiler will add a public default constructor, which will show up in the API.

Question 3

Question

Why do we add a private constructor to the `Converter` class?

Answer

Otherwise, the compiler will add a public default constructor, which will show up in the API.

Marking scheme

1 mark for mentioning the compiler, 1 mark for mentioning public constructor, 1 mark for mentioning API.

Question 4

Question

Assuming that `number >= 0 && number <= 1000` is its precondition, give a test vector for the `quarterToCents` method.

Question 4

Question

Assuming that `number >= 0 && number <= 1000` is its precondition, give a test vector for the `quarterToCents` method.

Answer

Boundary cases: 0 and 1000

Other cases: random/all integers in the interval [0, 1000].

Question 4

Question

Assuming that `number >= 0 && number <= 1000` is its precondition, give a test vector for the `quarterToCents` method.

Answer

Boundary cases: 0 and 1000
Other cases: random/all integers in the interval [0, 1000].

Marking scheme

1 mark for each boundary case, 1 mark for the random/all integers.

Question 5

Question

How do you document the following precondition?

```
number >= 0 && number <= 1000
```

Question 5

Question

How do you document the following precondition?

```
number >= 0 && number <= 1000
```

Answer

```
@pre. number >= 0 && number <= 1000
```

Question 5

Question

How do you document the following precondition?

```
number >= 0 && number <= 1000
```

Answer

```
@pre. number >= 0 && number <= 1000
```

Marking scheme

1 mark for @pre., 2 marks for the rest.

Question 6

Question

How many hours in total (including lecture time and lab time) did you spend on this course during the first week of the term?

Question 6

Question

How many hours in total (including lecture time and lab time) did you spend on this course during the first week of the term?

Marking scheme

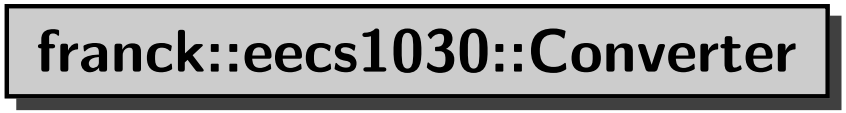
(the number of hours - 9) marks

Yes, your score can be negative here.

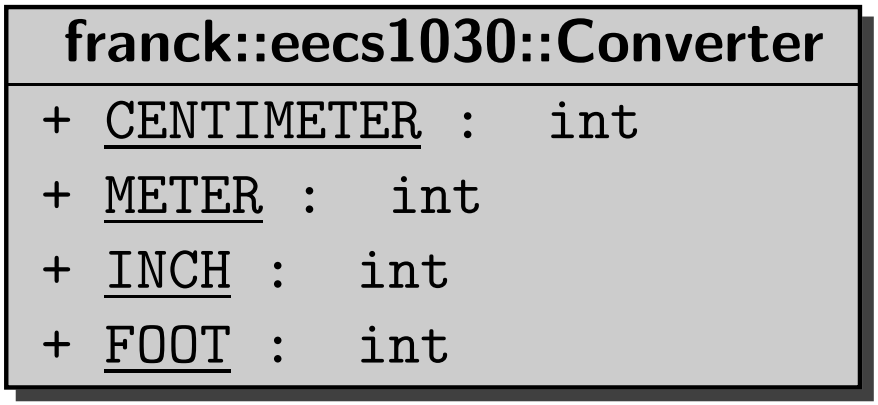
Total score

≥ 10	keep up the good work
< 5	please start studying before it is too late

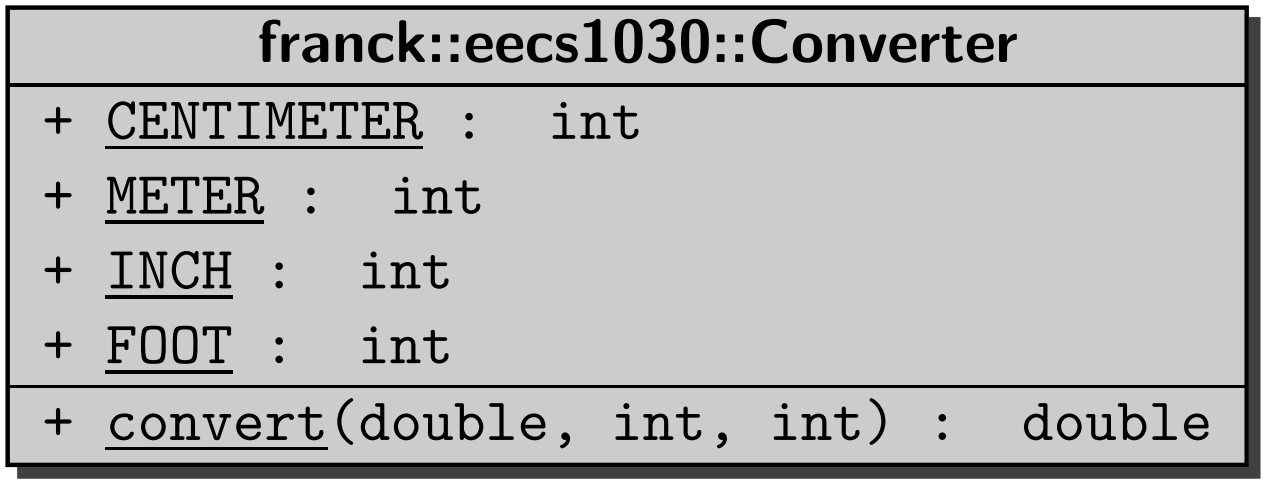
UML diagrams



UML diagrams



UML diagrams

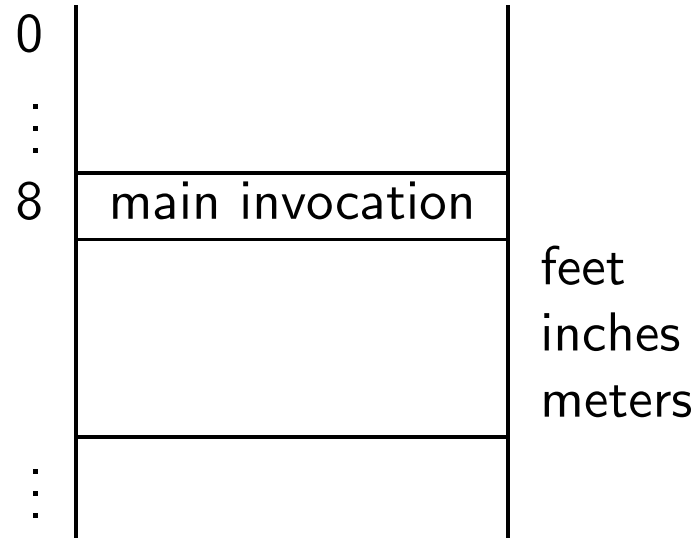


Converter

```
output.print("Number of feet: ");
int feet = input.nextInt();
output.print("Number of inches: ");
int inches = input.nextInt();
double meters =
  Converter.convert(feet, Converter.FOOT, Converter.METER)
+ Converter.convert(inches, Converter.INCH, Converter.METER);
output.printf("%d'%d = %.2f m", feet, inches, meters);
```

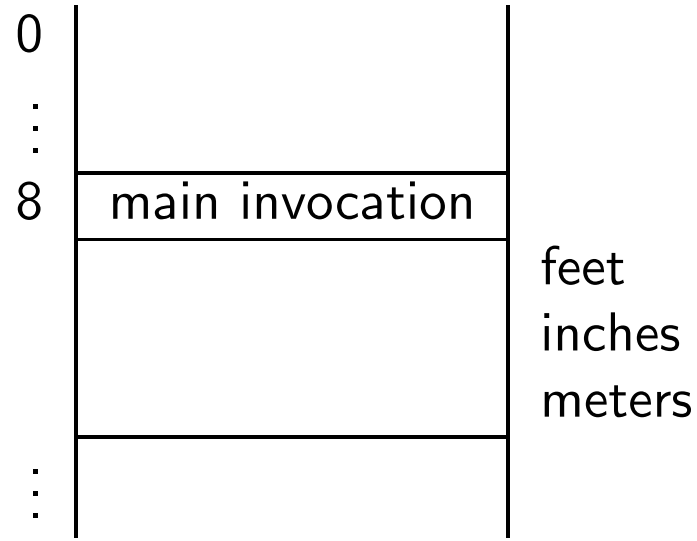
Memory diagrams

```
output.print("Number of feet: ");
int feet = input.nextInt();
output.print("Number of inches: ");
int inches = input.nextInt();
double meters =
   C.... convert(feet, C.... FOOT, C....METER)
+ C.... convert(inches, C.... INCH, C.... METER);
output.printf("%d'%d = %.2f m", feet, inches, meters);
```



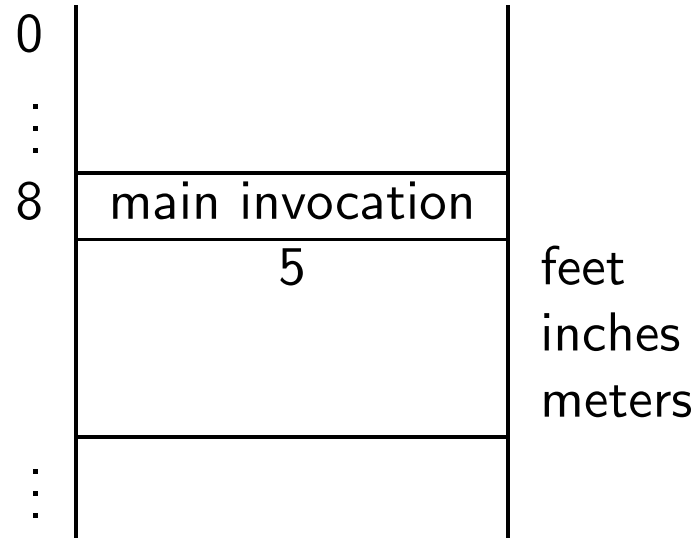
Memory diagrams

```
output.print("Number of feet: ");
int  feet  = input.nextInt();
output.print("Number of inches: ");
int  inches  = input.nextInt();
double meters =
   C.... convert( feet ,  C .... FOOT, C....METER)
+ C.... convert( inches ,  C .... INCH, C.... METER);
output. printf ("%d'%d = %.2f m", feet, inches, meters);
```



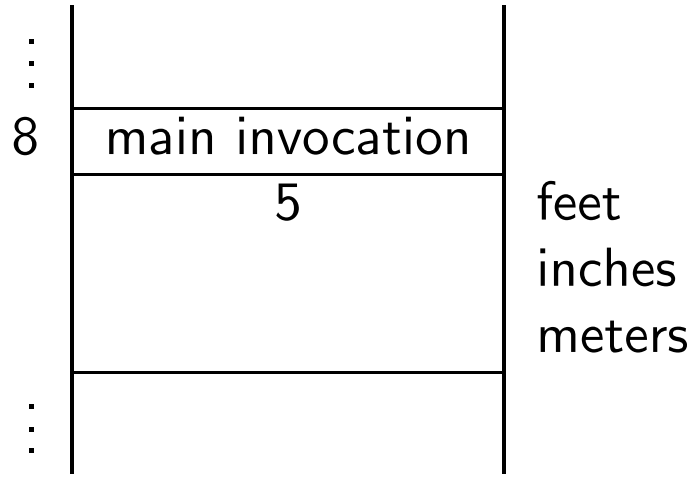
Memory diagrams

```
output.print("Number of feet: ");
int feet = input.nextInt();
output.print("Number of inches: ");
int inches = input.nextInt();
double meters =
   C.... convert(feet, C.... FOOT, C....METER)
+ C.... convert(inches, C.... INCH, C.... METER);
output.printf("%d'%d = %.2f m", feet, inches, meters);
```



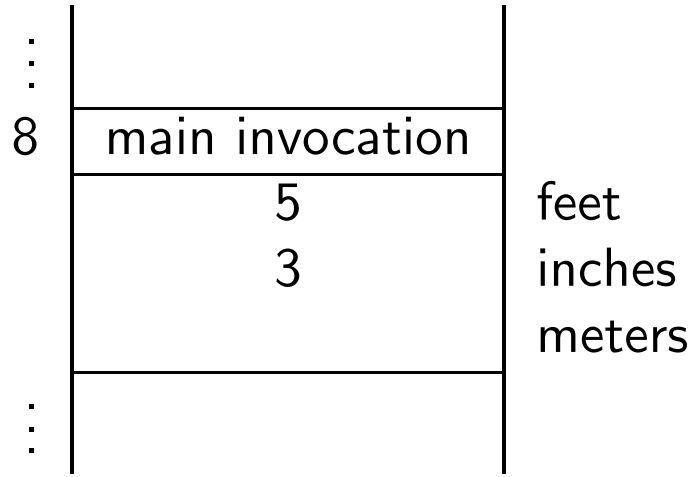
Memory diagrams

```
output.print("Number of feet: ");
int feet = input.nextInt();
output.print("Number of inches: ");
int inches = input.nextInt();
double meters =
   C.... convert(feet, C.... FOOT, C....METER)
+ C.... convert(inches, C.... INCH, C.... METER);
output.printf("%d'%d = %.2f m", feet, inches, meters);
```



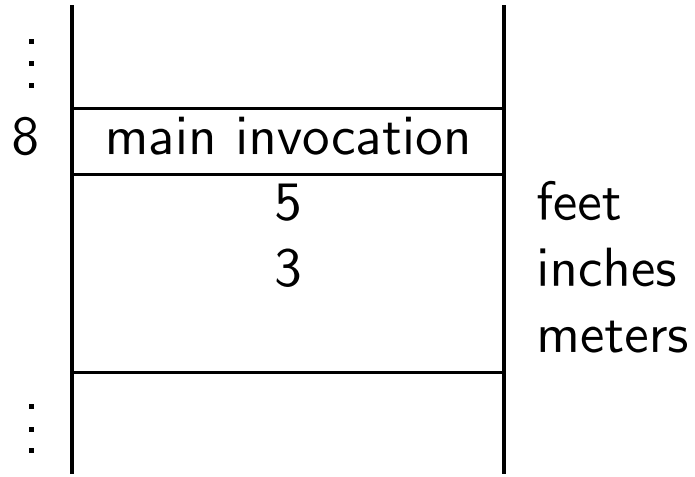
Memory diagrams

```
output.print("Number of feet: ");
int feet = input.nextInt();
output.print("Number of inches: ");
int inches = input.nextInt();
double meters =
   C.... convert(feet, C.... FOOT, C....METER)
+ C.... convert(inches, C.... INCH, C.... METER);
output.printf("%d'%d = %.2f m", feet, inches, meters);
```



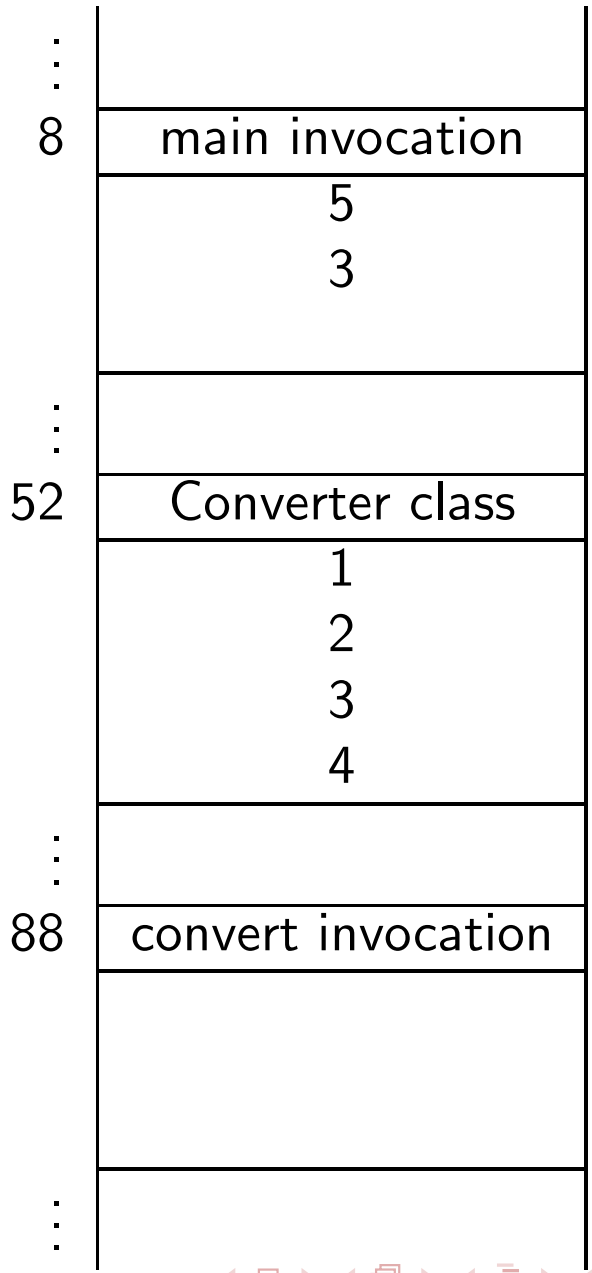
Memory diagrams

```
output.print("Number of feet: ");
int feet = input.nextInt();
output.print("Number of inches: ");
int inches = input.nextInt();
double meters =
  C....convert(feet, C....FOOT, C....METER)
+ C....convert(inches, C....INCH, C....METER);
output.printf("%d'%d = %.2f m", feet, inches, meters);
```



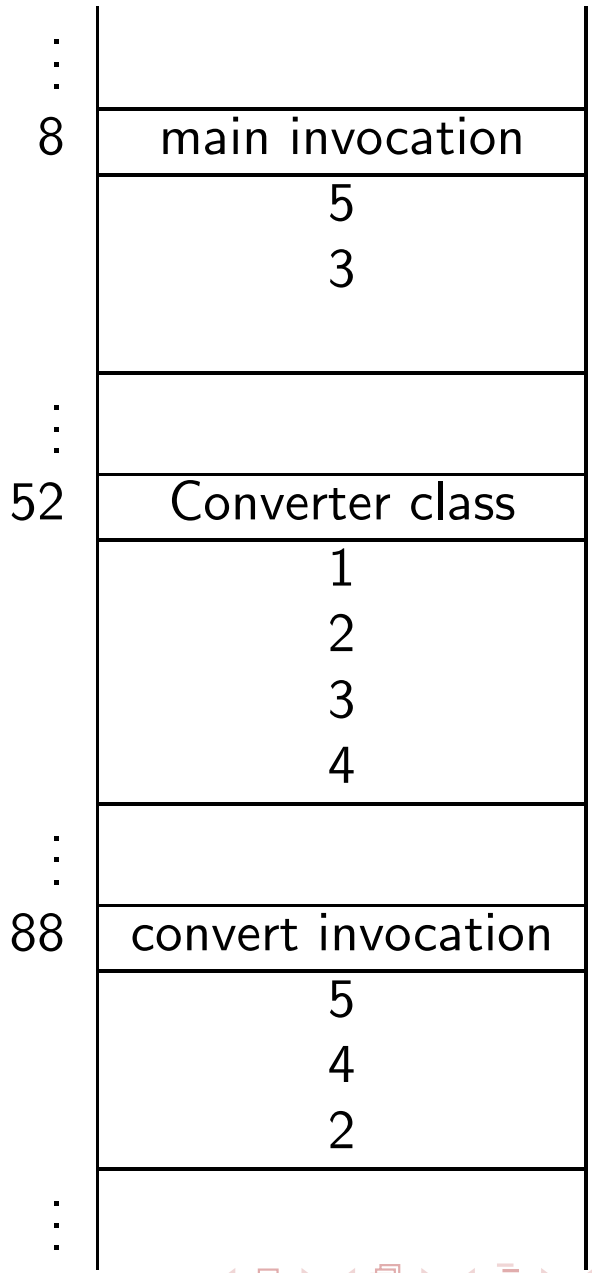
Memory diagrams

```
output.print("Number of feet: ");
int feet = input.nextInt();
output.print("Number of inches: ");
int inches = input.nextInt();
double meters =
  C....convert(feet, C....FOOT, C....METER)
+ C....convert(inches, C....INCH, C....METER);
output.printf("%d'%d = %.2f m", feet, inches, meters);
```



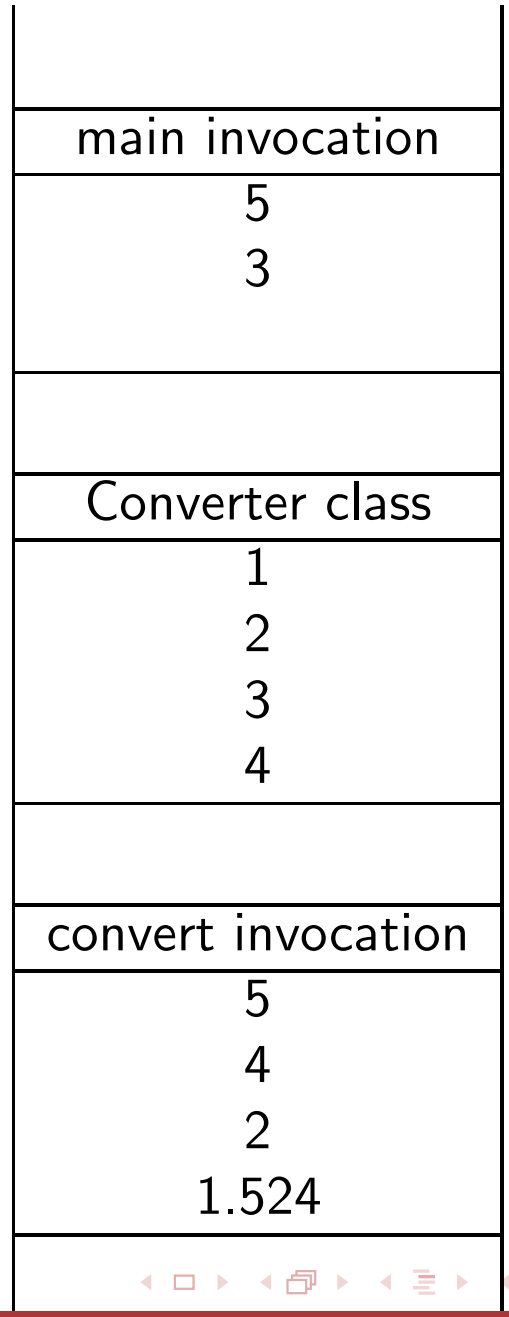
Memory diagrams

```
output.print("Number of feet: ");
int feet = input.nextInt();
output.print("Number of inches: ");
int inches = input.nextInt();
double meters =
  C....convert(feet, C....FOOT, C....METER)
+ C....convert(inches, C....INCH, C....METER);
output.printf("%d'%d = %.2f m", feet, inches, meters);
```



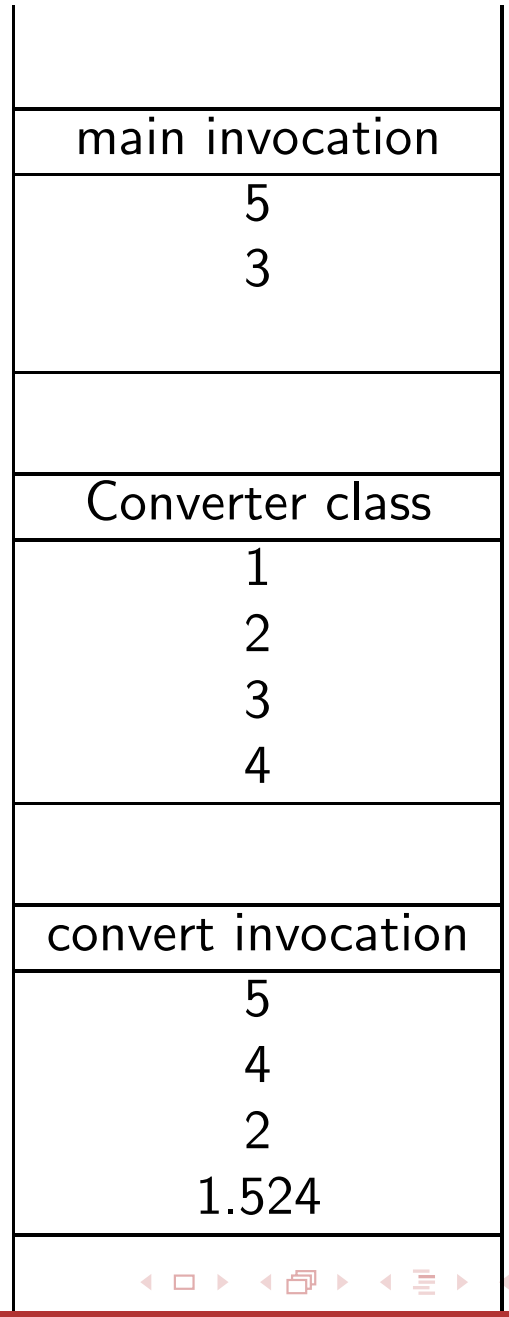
Memory diagrams

```
converted = amount *
   CENTIMETERS_PER_FOOT / CENTIMETERS_PER_METER;
return converted;
```



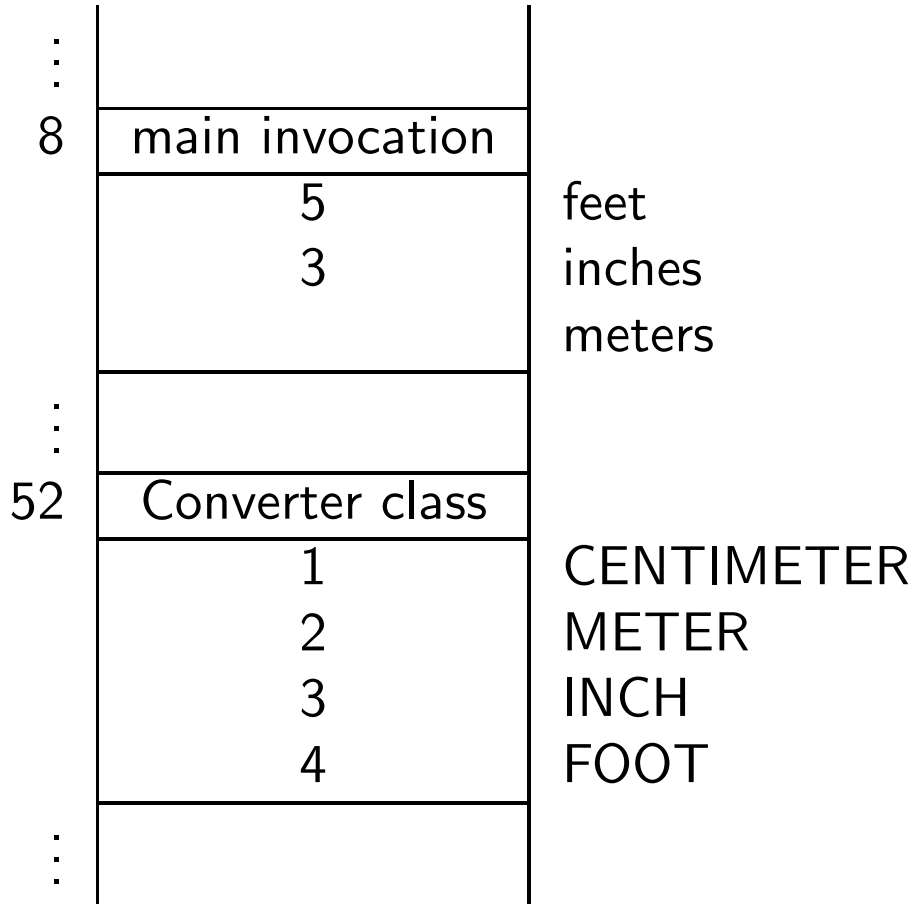
Memory diagrams

```
converted = amount *
   CENTIMETERS_PER_FOOT / CENTIMETERS_PER_METER;
return converted;
```



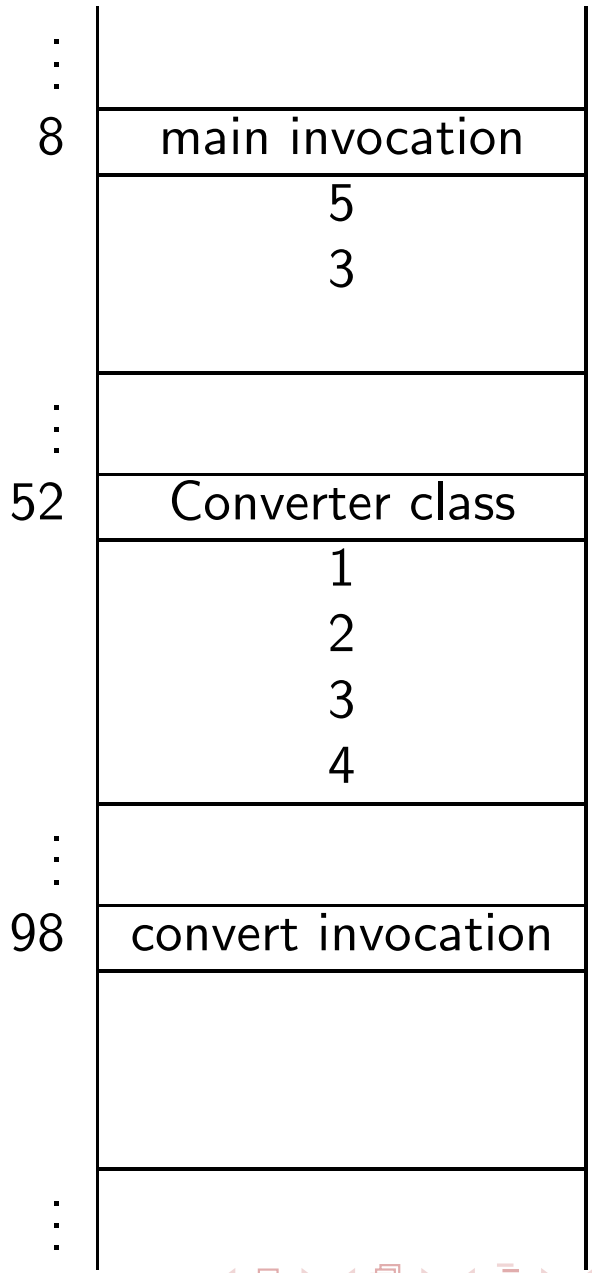
Memory diagrams

```
output.print("Number of feet: ");
int feet = input.nextInt();
output.print("Number of inches: ");
int inches = input.nextInt();
double meters =
  C.... convert(feet, C.... FOOT, C....METER) // 1.524
+ C....convert(inches, C....INCH, C....METER);
output.printf("%d'%d = %.2f m", feet, inches, meters);
```



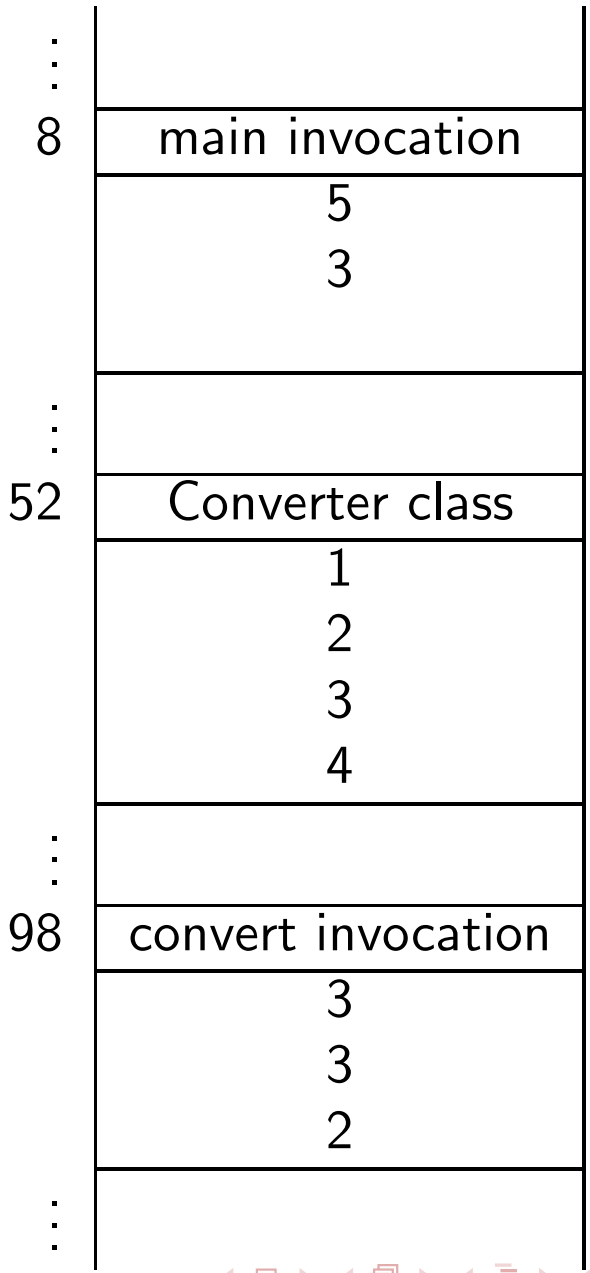
Memory diagrams

```
output.print("Number of feet: ");
int feet = input.nextInt();
output.print("Number of inches: ");
int inches = input.nextInt();
double meters =
  C.... convert(feet, C.... FOOT, C....METER) // 1.524
+ C....convert(inches, C....INCH, C....METER);
output.printf("%d'%d = %.2f m", feet, inches, meters);
```



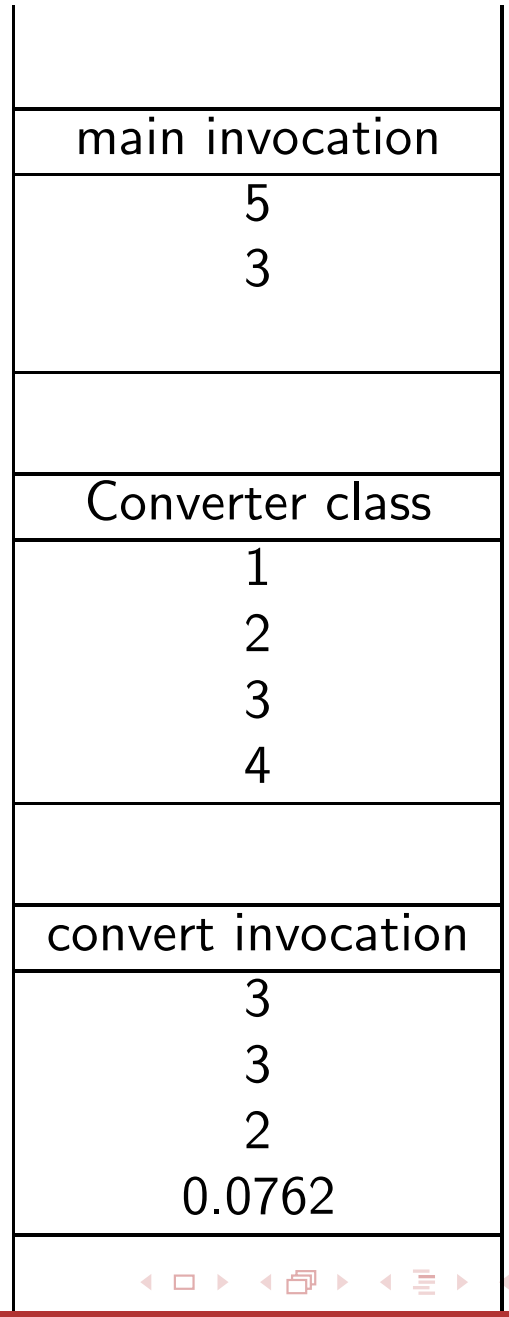
Memory diagrams

```
output.print("Number of feet: ");
int feet = input.nextInt();
output.print("Number of inches: ");
int inches = input.nextInt();
double meters =
  C.... convert(feet, C.... FOOT, C....METER) // 1.524
+ C....convert(inches, C....INCH, C....METER);
output.printf("%d'%d = %.2f m", feet, inches, meters);
```



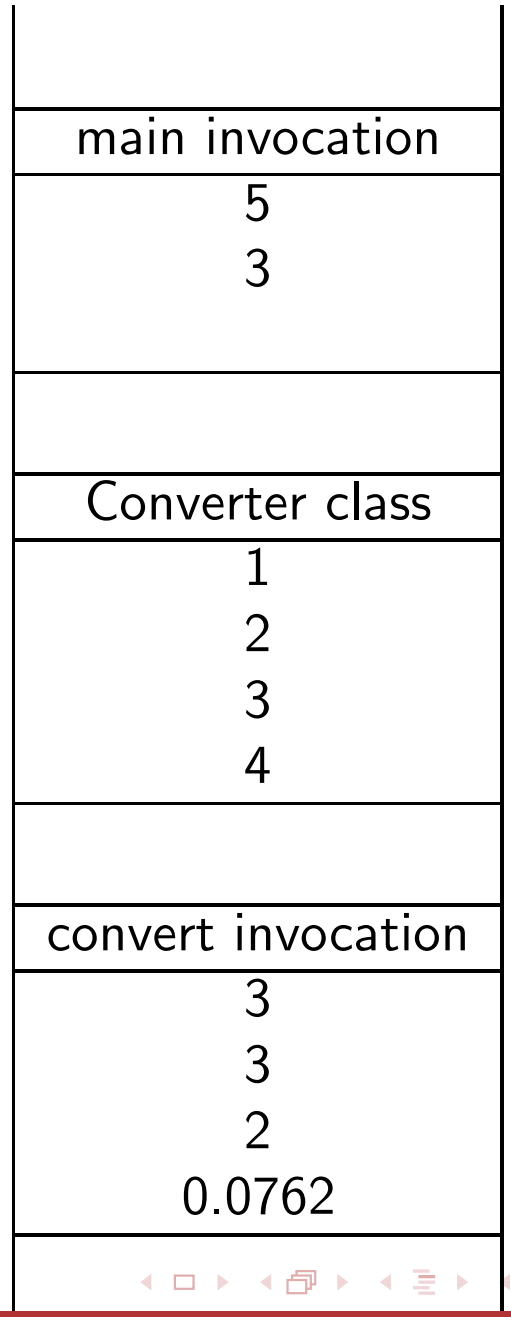
Memory diagrams

```
converted = amount *
   CENTIMETERS_PER_INCH / CENTIMETERS_PER_METER;
return converted;
```



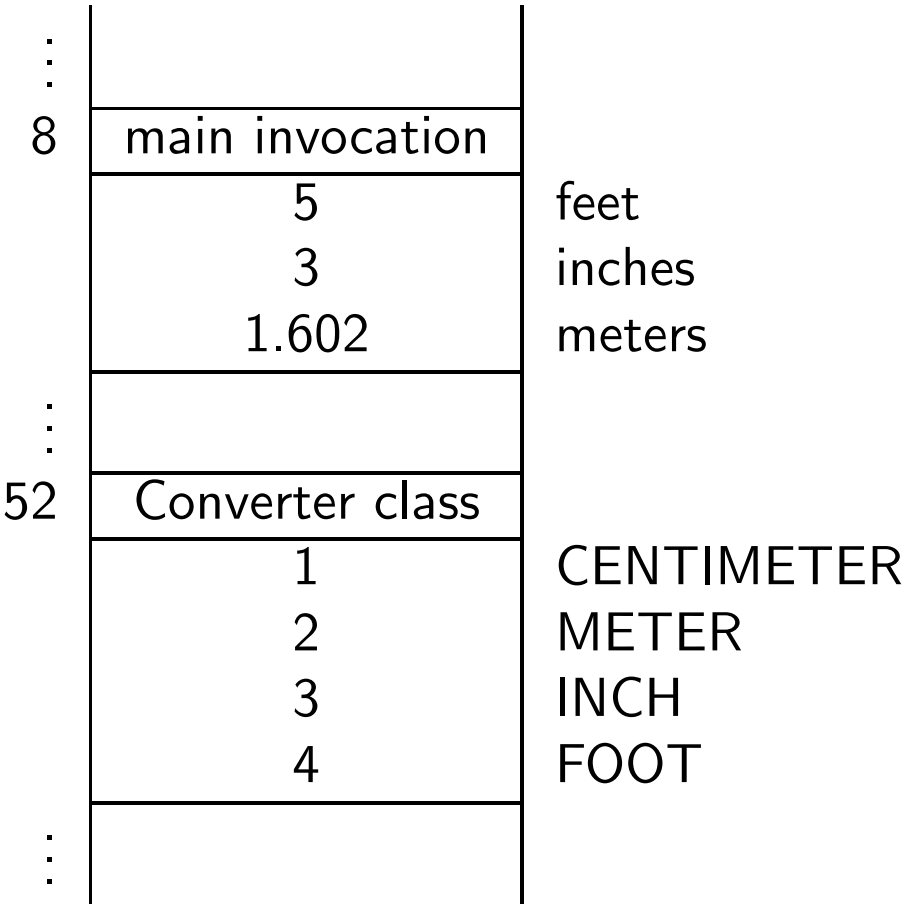
Memory diagrams

```
converted = amount *
   CENTIMETERS_PER_INCH / CENTIMETERS_PER_METER;
return converted;
```



Memory diagrams

```
output.print("Number of feet: ");
int feet = input.nextInt();
output.print("Number of inches: ");
int inches = input.nextInt();
double meters =
   C.... convert(feet, C.... FOOT, C....METER) // 1.524
+ C.... convert(inches, C.... INCH, C.... METER); // 0.0762
output.printf("%d'%d = %.2f m", feet, inches, meters);
```



Cheapest, fastest, earliest, . . .

Problem (specification)

Given a `list` and an `element`, is the `element` smaller than all the elements in the `list`?

Cheapest, fastest, earliest, . . .

Design

Do this on a piece of paper.

Cheapest, fastest, earliest, . . .

Design

Do this on a piece of paper.

```
smaller = ?
for each element e of list
  smaller = (element is smaller than e) ? smaller
```

Cheapest, fastest, earliest, . . .

```
smaller = ?
for each element e of list
  smaller = (element is smaller than e) ? smaller
```

Question

Fill in the ?s.

Cheapest, fastest, earliest, . . .

```
smaller = ?
for each element e of list
  smaller = (element is smaller than e) ? smaller
```

Question

Fill in the ?s.

Answer

```
smaller = true
for each element e of list
  smaller = (element is smaller than e) && smaller
```

Cheapest, fastest, earliest, . . .

```
smaller = true
for each element e of list
  smaller = (element is smaller than e) && smaller
```

Cheapest, fastest, earliest, . . .

```
smaller = true
for each element e of list
  smaller = (element is smaller than e) && smaller
```

Implementation

Implement the above pseudocode in Java.

Cheapest, fastest, earliest, . . .

```
smaller = true
for each element e of list
  smaller = (element is smaller than e) && smaller
```

Implementation

Implement the above pseudocode in Java.

Answer

This can be implemented in many different ways, including

```
boolean smaller = true;
int i = 0;
while (i < list.size())
{
   smaller = element.compareTo(list.get(i)) < 0 && smaller
   i++;
}
```

Cheapest, fastest, earliest, . . .

Question

How can we argue that the following code is correct?

```
boolean smaller = true;
int i = 0;
while (i < list.size())
{
   smaller = element.compareTo(list.get(i)) < 0 && smaller
   i++;
}
```

Cheapest, fastest, earliest, . . .

Question

How can we argue that the following code is correct?

```
boolean smaller = true;
int i = 0;
while (i < list.size())
{
   smaller = element.compareTo(list.get(i)) < 0 && smaller
   i++;
}
```

Answer

By introducing a loop invariant.

Loop Invariant

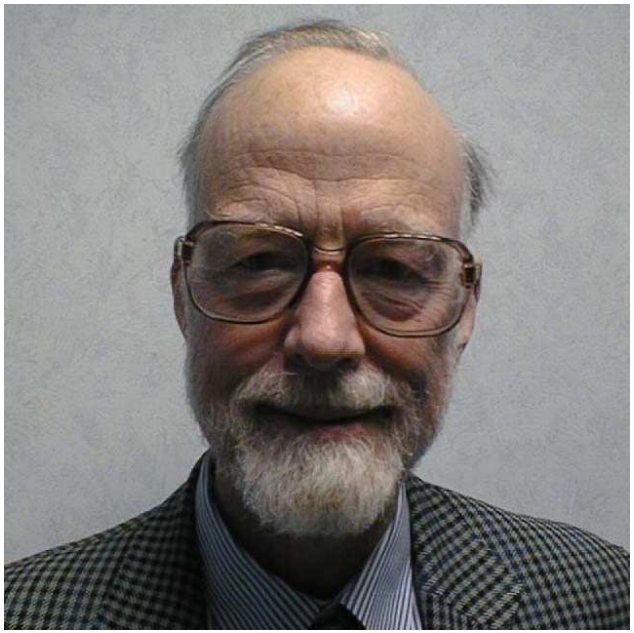
Definition

Given a loop, a Boolean expression is a *loop invariant* of the loop if it holds at the beginning of every iteration of the loop.

C.A.R. Hoare. An Axiomatic Basis for Computer Programming. *Communications of the ACM*, 12(10): 576–580, October 1969.

Tony Hoare

Sir Charles Antony Richard Hoare (born January 11, 1934) is a British computer scientist. He is best known for the development of Quicksort, an algorithm to sort elements. He also proposed an approach to reason about loops. In 1980, he received the Turing award.



source: research.microsoft.com

Loop invariant

```
boolean smaller = true;
int i = 0;
/* LI */ while (i < list.size())
{
    smaller = element.compareTo(list.get(i)) < 0 && smaller
    i++;
}
```

Question

Is true a loop invariant for the above loop?

Loop invariant

```
boolean smaller = true;
int i = 0;
/* LI */ while (i < list.size())
{
    smaller = element.compareTo(list.get(i)) < 0 && smaller
    i++;
}
```

Question

Is true a loop invariant for the above loop?

Answer

Yes.

Loop invariant

```
boolean smaller = true;
int i = 0;
/* LI */ while (i < list.size())
{
    smaller = element.compareTo(list.get(i)) < 0 && smaller
    i++;
}
```

Question

Is `i >= 0` a loop invariant for the above loop?

Loop invariant

```
boolean smaller = true;
int i = 0;
/* LI */ while (i < list.size())
{
    smaller = element.compareTo(list.get(i)) < 0 && smaller
    i++;
}
```

Question

Is `i >= 0` a loop invariant for the above loop?

Answer

Yes.

Loop invariant

```
boolean smaller = true;
int i = 0;
/* LI */ while (i < list.size())
{
    smaller = element.compareTo(list.get(i)) < 0 && smaller
    i++;
}
```

Question

Is `i < list.size()` a loop invariant for the above loop?

Loop invariant

```
boolean smaller = true;
int i = 0;
/* LI */ while (i < list.size())
{
    smaller = element.compareTo(list.get(i)) < 0 && smaller
    i++;
}
```

Question

Is `i < list.size()` a loop invariant for the above loop?

Answer

No, the last time that the beginning of the loop is reached, we have that `i == list.size()`.

Loop invariant

```
boolean smaller = true;
int i = 0;
/* LI */ while (i < list.size())
{
    smaller = element.compareTo(list.get(i)) < 0 && smaller
    i++;
}
```

Question

Is `i <= list.size()` a loop invariant for the above loop?

Loop invariant

```
boolean smaller = true;
int i = 0;
/* LI */ while (i < list.size())
{
    smaller = element.compareTo(list.get(i)) < 0 && smaller
    i++;
}
```

Question

Is `i <= list.size()` a loop invariant for the above loop?

Answer

Yes.

Loop invariant

```
boolean smaller = true;
int i = 0;
/* LI */ while (i < list.size())
{
    smaller = element.compareTo(list.get(i)) < 0 && smaller
    i++;
}
```

Question

Is `i >= 0 && i <= list.size()` a loop invariant for the above loop?

Loop invariant

```
boolean smaller = true;
int i = 0;
/* LI */ while (i < list.size())
{
    smaller = element.compareTo(list.get(i)) < 0 && smaller
    i++;
}
```

Question

Is `i >= 0 && i <= list.size()` a loop invariant for the above loop?

Answer

Yes.

Loop invariant

```
boolean smaller = true;
int i = 0;
/* LI */ while (i < list.size())
{
    smaller = element.compareTo(list.get(i)) < 0 && smaller
    i++;
}
```

Question

Is `smaller == element.compareTo(list.get(0)) < 0 && ... && element.compareTo(list.get(i - 1)) < 0` a loop invariant for the above loop?

Loop invariant

```
boolean smaller = true;
int i = 0;
/* LI */ while (i < list.size())
{
    smaller = element.compareTo(list.get(i)) < 0 && smaller
    i++;
}
```

Question

Is `smaller == element.compareTo(list.get(0)) < 0 && ... && element.compareTo(list.get(i - 1)) < 0` a loop invariant for the above loop?

Answer

Yes.

Loop invariant

```
boolean smaller = true;
int i = 0;
/* LI */ while (i < list.size())
{
    smaller = element.compareTo(list.get(i)) < 0 && smaller
    i++;
}
```

Question

Is `i >= 0 && i <= list.size() && smaller == element.compareTo(list.get(0)) < 0 && ... && element.compareTo(list.get(i - 1)) < 0` a loop invariant for the above loop?

Loop invariant

```
boolean smaller = true;
int i = 0;
/* LI */ while (i < list.size())
{
    smaller = element.compareTo(list.get(i)) < 0 && smaller
    i++;
}
```

Question

Is `i >= 0 && i <= list.size() && smaller == element.compareTo(list.get(0)) < 0 && ... && element.compareTo(list.get(i - 1)) < 0` a loop invariant for the above loop?

Answer

Yes.

Cheapest, fastest, earliest, . . .

```
boolean smaller = true;
int i = 0;
while (i < list.size())
{
   smaller = element.compareTo(list.get(i)) < 0 && smaller;
   i++;
}
return smaller;
```

Question

Is the method header

```
public static boolean smaller(List<Integer> list,
                              Integer element)
```

appropriate?

Cheapest, fastest, earliest, . . .

```
boolean smaller = true;
int i = 0;
while (i < list.size())
{
   smaller = element.compareTo(list.get(i)) < 0 && smaller;
   i++;
}
return smaller;
```

Question

Is the method header

```
public static boolean smaller(List<Integer> list,
                              Integer element)
```

appropriate?

Answer

Yes, because the class `Integer` implements the interface `Comparable<Integer>` and, hence, implements the method `compareTo(Integer)`.

Cheapest, fastest, earliest, . . .

```
boolean smaller = true;
int i = 0;
while (i < list.size())
{
   smaller = element.compareTo(list.get(i)) < 0 && smaller;
   i++;
}
return smaller;
```

Question

Is the method header

```
public static boolean smaller(List<Double> list,
                              Double element)
```

appropriate?

Cheapest, fastest, earliest, . . .

```
boolean smaller = true;
int i = 0;
while (i < list.size())
{
   smaller = element.compareTo(list.get(i)) < 0 && smaller;
   i++;
}
return smaller;
```

Question

Is the method header

```
public static boolean smaller(List<Double> list,
                              Double element)
```

appropriate?

Answer

Yes, because class `Double` implements the interface `Comparable<Double>` and, hence, implements the method `compareTo(Double)`.

Cheapest, fastest, earliest, . . .

```
boolean smaller = true;
int i = 0;
while (i < list.size())
{
   smaller = element.compareTo(list.get(i)) < 0 && smaller;
   i++;
}
return smaller;
```

Question

Is the method header

```
public static boolean smaller(List<Date> list,
                              Date element)
```

appropriate?

Cheapest, fastest, earliest, . . .

```
boolean smaller = true;
int i = 0;
while (i < list.size())
{
   smaller = element.compareTo(list.get(i)) < 0 && smaller;
   i++;
}
return smaller;
```

Question

Is the method header

```
public static boolean smaller(List<Date> list,
                              Date element)
```

appropriate?

Answer

Yes, because class `Date` implements the interface `Comparable<Date>` and, hence, implements the method `compareTo(Date)`.

Cheapest, fastest, earliest, . . .

```
boolean smaller = true;
int i = 0;
while (i < list.size())
{
   smaller = element.compareTo(list.get(i)) < 0 && smaller;
   i++;
}
return smaller;
```

Question

Can we introduce one method header that captures all previous cases, that is, one that accepts lists of Integers, Doubles, and Dates?

Cheapest, fastest, earliest, . . .

```
boolean smaller = true;
int i = 0;
while (i < list.size())
{
   smaller = element.compareTo(list.get(i)) < 0 && smaller;
   i++;
}
return smaller;
```

Question

Can we introduce one method header that captures all previous cases, that is, one that accepts lists of Integers, Doubles, and Dates?

Answer

Yes, if we use generics.

```
boolean smaller = true;
int i = 0;
while (i < list.size())
{
   smaller = element.compareTo(list.get(i)) < 0 && smaller;
   i++;
}
return smaller;
```

Question

Is the method header

```
public static <T> boolean smaller(List<T> list, T element)
```

appropriate?

Generics

```
boolean smaller = true;
int i = 0;
while (i < list.size())
{
   smaller = element.compareTo(list.get(i)) < 0 && smaller;
   i++;
}
return smaller;
```

Question

Is the method header

`public static <T> boolean smaller(List<T> list, T element)`

appropriate?

Answer

No, because `T` may not implement the interface `Comparable<T>`.

```
boolean smaller = true;
int i = 0;
while (i < list.size())
{
   smaller = element.compareTo(list.get(i)) < 0 && smaller;
   i++;
}
return smaller;
```

Question

How can we specify that `T` implements the interface `Comparable<T>`?

```
boolean smaller = true;
int i = 0;
while (i < list.size())
{
   smaller = element.compareTo(list.get(i)) < 0 && smaller;
   i++;
}
return smaller;
```

Question

How can we specify that T implements the interface Comparable<T>?

Answer

```
public static <T extends Comparable<T>> boolean
   smaller(List<T> list, T element)
```