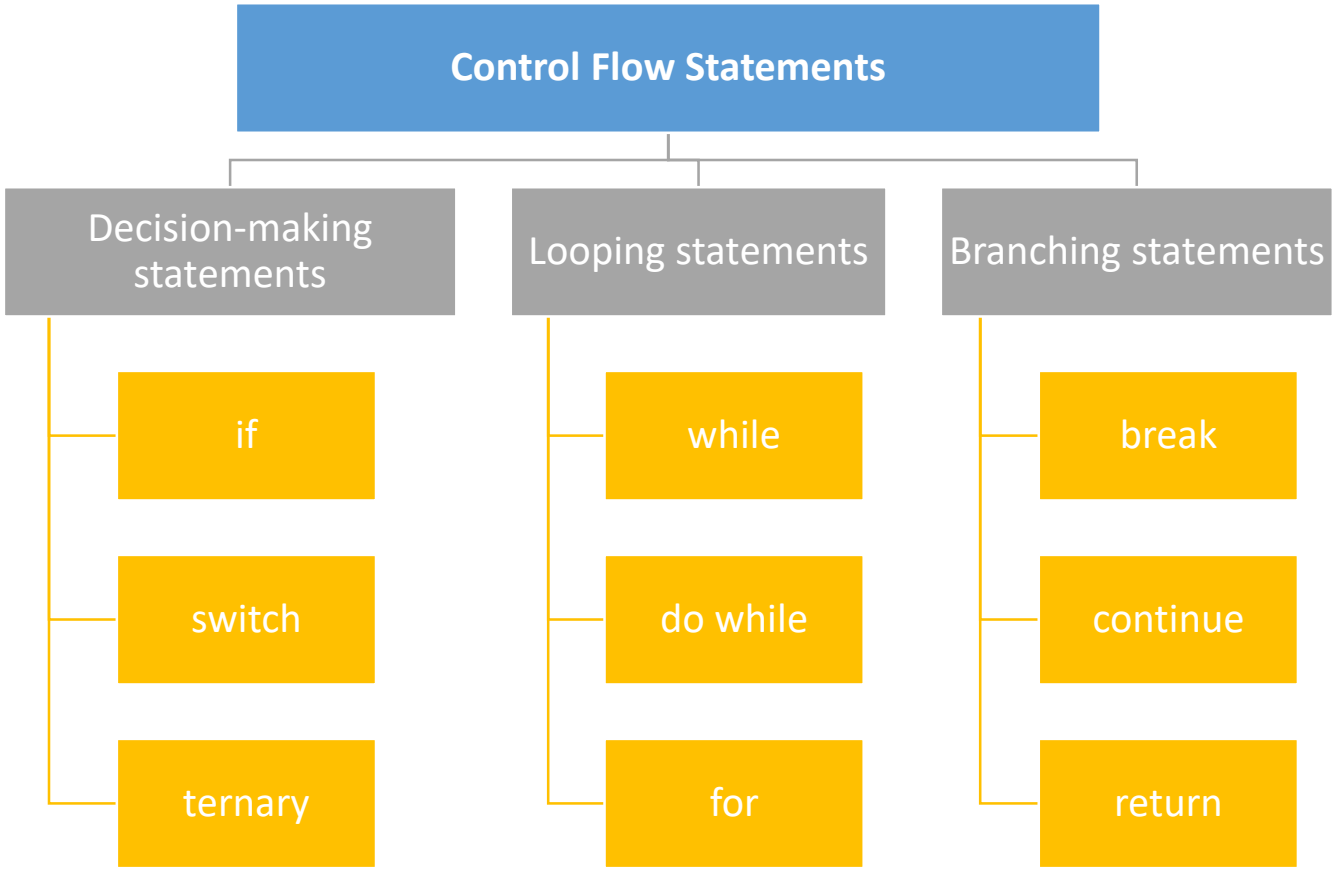
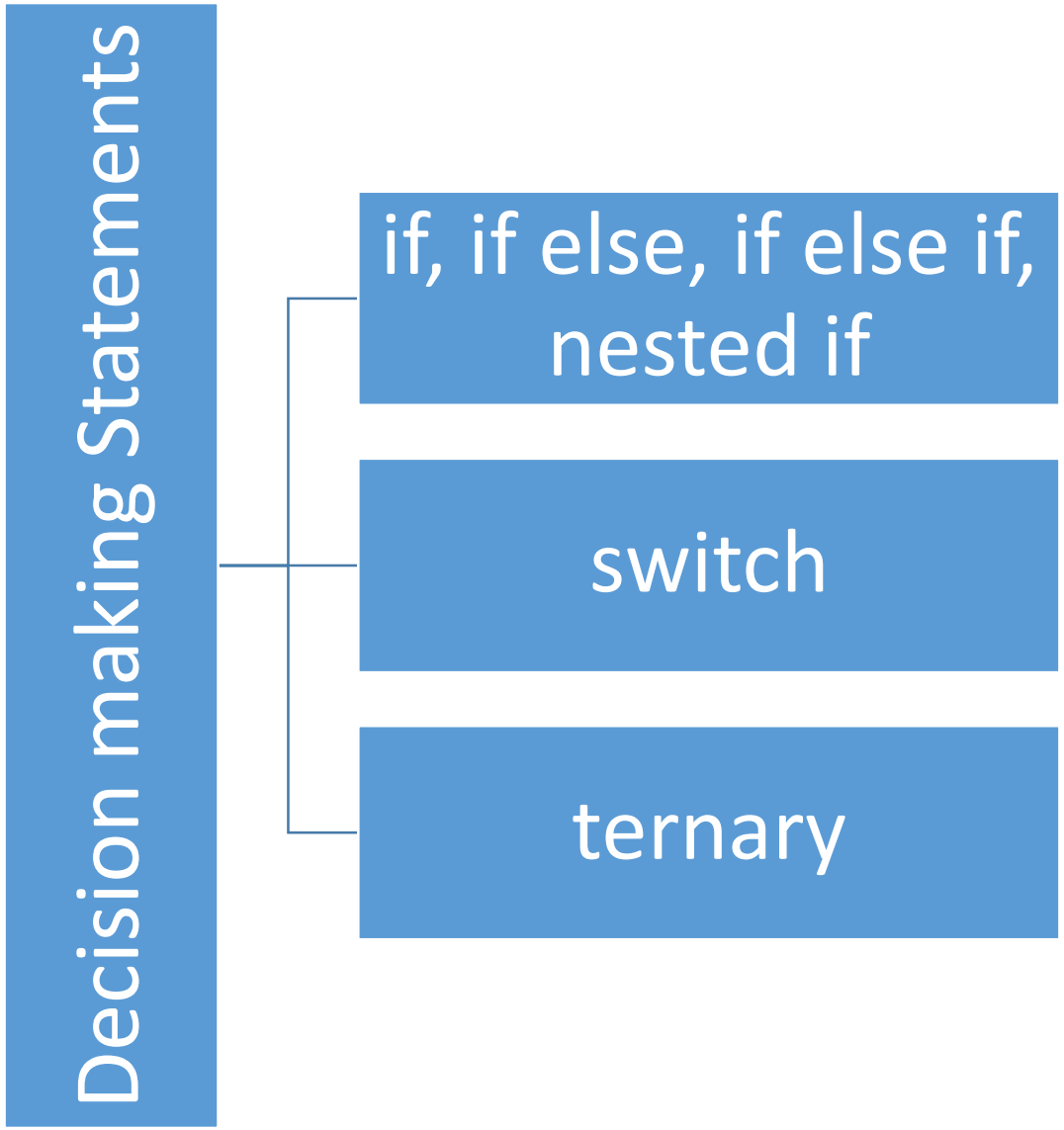

Control Flow Statements
Decision-making statements
Looping statements
Branching statements
if
switch
ternary
while
do while
for
break
continue
return


Decision making Statements
if, if else, if else if, nested if
switch
ternary

if-then statement is the most basic of all the control flow statements. It tells your program to execute a certain section of code only if a particular test evaluates to true.

If else

Syntax:

```
if(condition)
{
Statements if condition is true
}
else
{
Statements if condition is false
}
```

If else if

Syntax:

```
if(condition1)
{
Statements if condition1 is true
}
else if(condition2)
{
Statements if condition2 is false
}
if(condition1)
{
Statements if condition3 is true
}
.
.
else
{
Statements if condition is false
}
```

if

Syntax:

```
if(condition)
{
Statements if condition is true
}
```

Nested if

It is possible to have if / if else / if else if inside another if / if else /if else if .

Switch statement is a type of selection control mechanism used to allow the value of a variable or expression to change the control flow of program. Switch case statements are a substitute for long if statements that compare a variable to several "integral" values

switch

Syntax:

```
switch(expression/condition)
{
        case label1 : statements;
                      break;
        case label2 : statements;
                      break;
        case label3:statements;
                      break;
        .
        .
        .

        default : statements;
}
```

A ternary operator is an operator that takes three arguments. It returns one of two expressions depending on a condition.

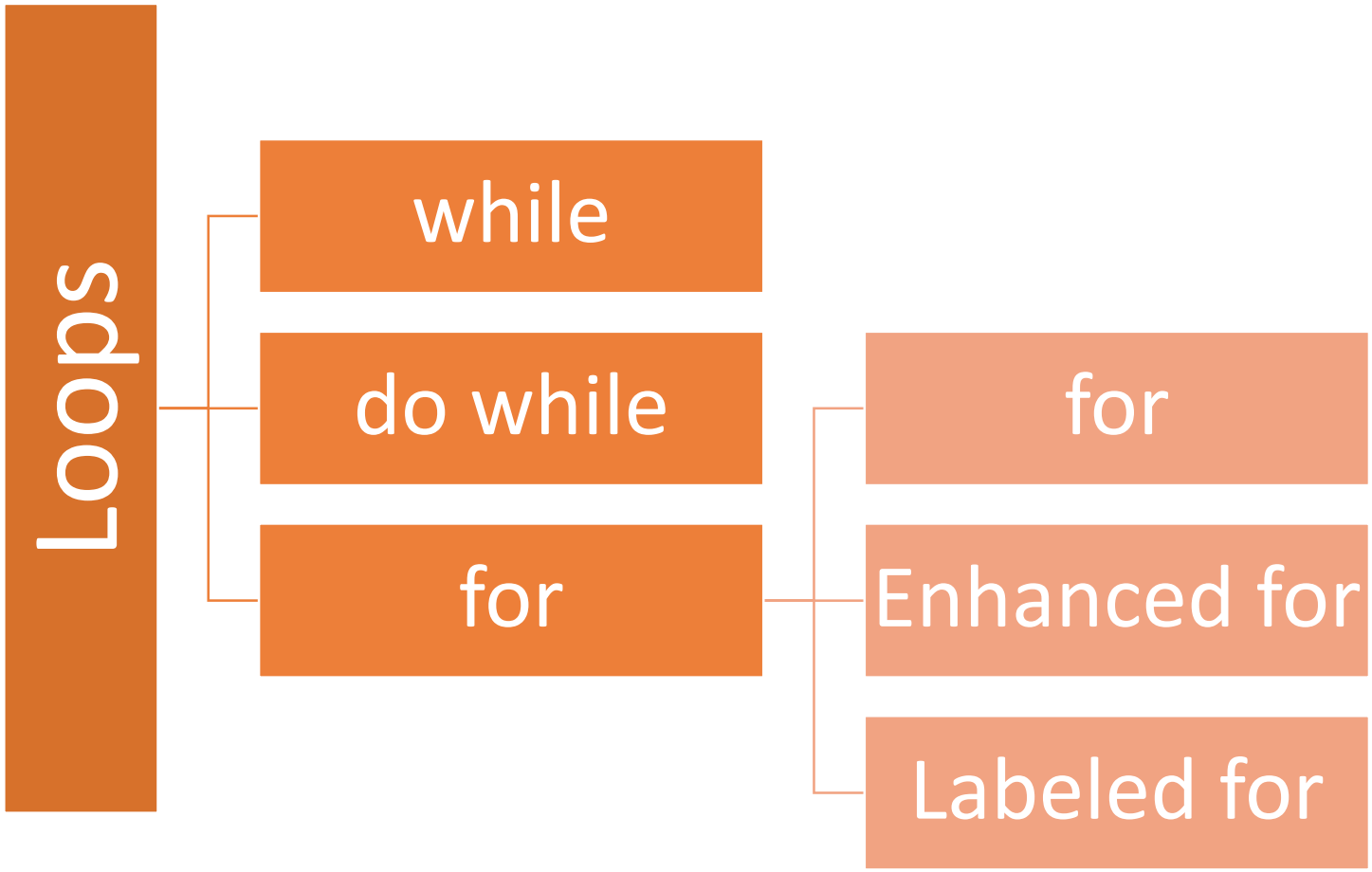
The ternary operator can return a value of any data type. Ternary operator can also be used instead of if-else statement.

Note:

- Ternary operator returns a value, it does not execute it.
- It can be used to replace a short if-else statement.
- A nested ternary operator is allowed. It will be evaluated from right to left

Ternary Operator (?:)

condition? statement to execute if condition is true : statement to execute if condition is false


Loops
while
do while
for
for
Enhanced for
Labeled for

A loop statements allows to execute statements or set of statements multiple times, where statements are executed sequentially. An infinite loop is one that lacks a functioning exit routine . The result is that the loop repeats continually until the operating system senses it and terminates the program with an error or until some other event occurs (such as having the program automatically terminate after a certain duration of time).

while

Syntax:

```
while(condition)
{
statements;
}
```

Notes:
while loop is entry controlled loop.

do while

Syntax:

```
do
{
statements;
}while(condition);
```

Notes:
do while is exit controlled loop. Statements are executed at least one time.

for loop

Syntax:

```
for(initialization; condition; updation)
{
statements
}
```

Note:
Initialization statement(s) execute only for the first time.

Enhanced for

Syntax:

```
for(type  variable : array/collection)
{
Statements
}
```

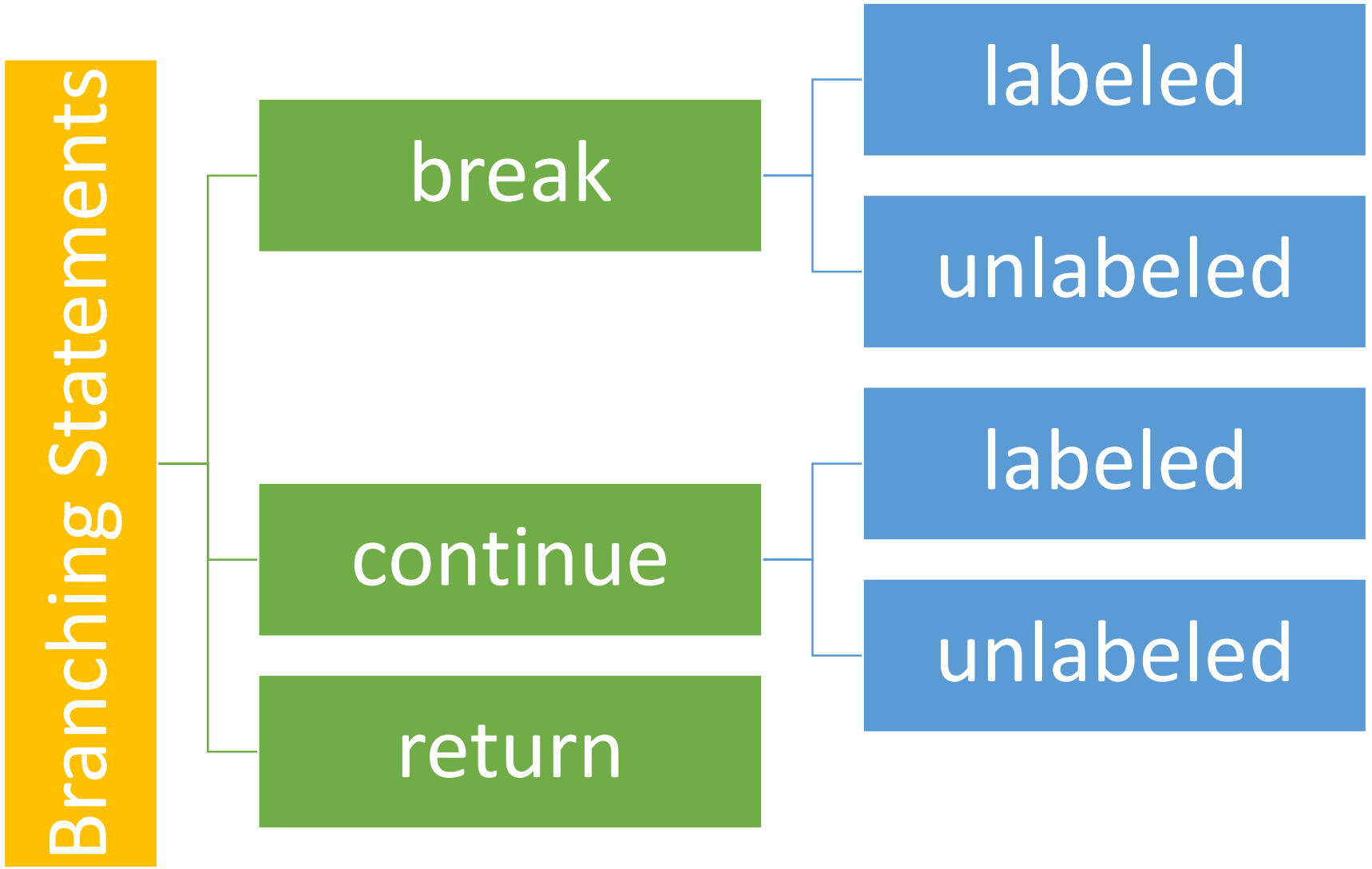
Notes:
Enhanced for loop is used to iterate through arrays and collections.

Labeled for

Syntax:

```
Labelname:
for(initialization; condition; updation)
{
statements
}
```

Note:
Labels are the name of the for loop. It is useful if we have nested for loop so that we can break/continue specific for loop.


Branching Statements
break
labeled
unlabeled
continue
labeled
unlabeled
return

Break

Unlabeled break

Syntax:
break;

Notes:
Used in loops and switch. ‘break’ break the control flow of the program without executing the rest of the statements inside the loop

Labeled break

Syntax:
break label_name;

Notes:
Labeled break terminates an outer block. Control flow is transferred to the statements following the labeled statements.

Continue

Unlabeled continue

Syntax:
continue;

Notes:
Continue statements skip the code right after it. For nested loop it continues the inner loop.

Labeled continue

Syntax:
continue label_name;

Notes:
A labeled continue statements skips the current iteration of the outer loop.

Return

Syntax:
return value;
or
return;

Notes:
The return statements exits from the current method, and control flow returns to where the methods was invoked. When a method is declared as void use the form of return that doesn’t return a value