Type Casting

Data Type	Size	Description
byte	1 byte	Whole numbers (-128 to 127)
short	2 bytes	Whole numbers (-32,768 to 32,767)
int	4 bytes	Whole numbers (-2,147,483,648 to 2,147,483,647)
long	8 bytes	Whole numbers (-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807)
float	4 bytes	Fractional numbers (6 to 7 decimal digits)
double	8 bytes	Fractional numbers (15 decimal digits)
boolean	1 bit	True or False
char	2 bytes	Single character

Type Casting

- Converting a value from one data type to another is called Type Casting
- Widening or Automatic Type or Implicit Conversion:
  - Converting a lower data type into higher one is called widening type casting
  - Two data types are converted automatically
  - Two data types are compatible
  - Assigning value of a smaller data type to bigger data type
  - byte -> short -> int -> long -> float -> double

Example:

```
int a = 10;
long b = a;
float c = a;
```

Type Casting

- Narrowing or Explicit Conversion:
  - Assigning a value of larger data type to a smaller data type is called explicit type casting
  - Used for incompatible data types
  - double -> float -> long -> int -> short -> byte

Example:

Double a = 20.1;

Long b = (long) a;

t c = (int) b;

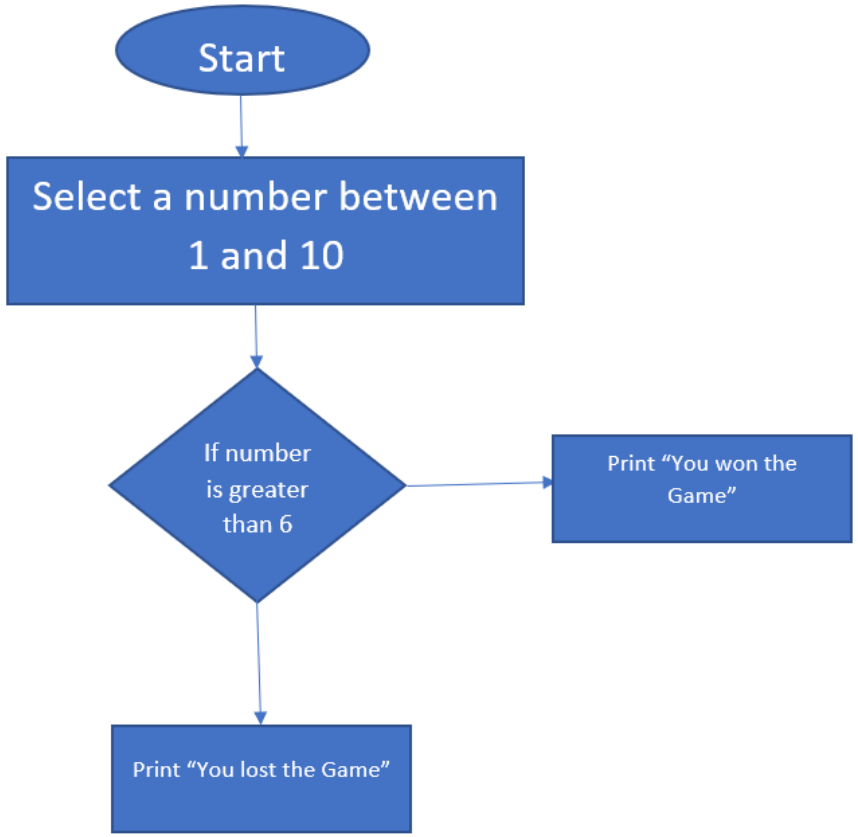
Control Flows

- A program's control flow is the order in which the program's code is executed
- Code execution flow can be changed by including control flow statements and conditions
- Conditions determine how many times a line of code gets executed
- Java provides 3 types of control flow statements:
  - Decision making statements – If, if-else, else-if
  - Loop statements – while, do while, for
  - Jump statements – break, continue

If Else Statements

Let’s map out the control flow of a Dice game program

- Player will select a number by rolling the dice
- If the number is greater than 6 , player will win or else will loose



QASCRIPT

If Else Statement

- An if statement is a control flow which performs some action when condition is true

  Condition: number > 6

  Action: print “You won the game” (Runs when condition is true)

- An else statement only runs when the condition is false

  Condition: number < 6

  Action: print “You lost the game” (Runs when condition is false)

QS
QASCRIPT

While Loop

Loop executes a block of code repeatedly based on the condition

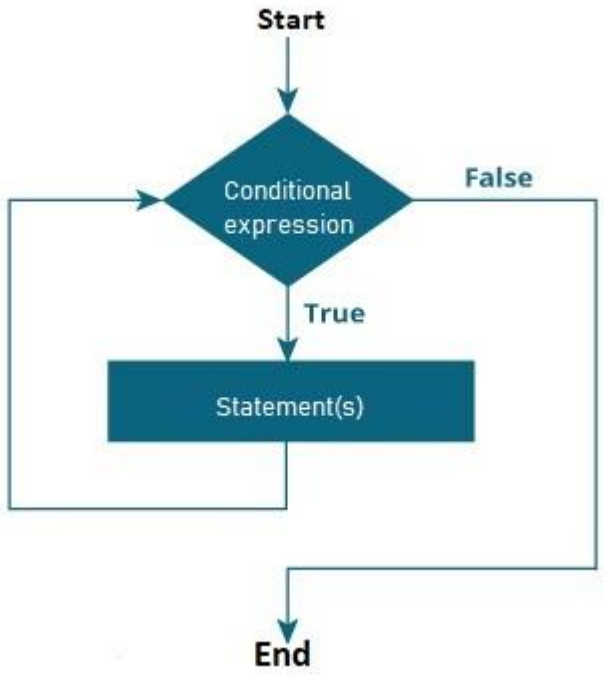
While Loop: It is a control flow statement that

allows code to be executed as long as a given

condition is true

Syntax:

```
while(condition){
    statement(s)
}
```



QASCRIPT

While Loop

Lets consider a music player where you can repeat songs with a while loop

Condition: Is Repeat button switched on

True: Print “Playing the same playlist again”

False: Print “Play the next playlist”

QS
QASCRIPT

Do While Loop

It is similar to while loop but checks the condition after the statements are executed

Syntax:

```
do{
        statement(s)
}
while(condition);
```

- All statements are executed until the condition is false and loop terminates
- It will execute the statement at least once

For Loop

For statement provides a compact way to iterate over a range of values. It repeatedly loops until a particular condition is satisfied.

Syntax:

```
for(initialization condition;termination condition;increment/decrement){
        statements
}
```

- Initialization expression initializes the loop, it is executed once as loop begins
- Termination expression evaluates to false, the loop terminates
- Increment expression is invoked after each iteration through the loop

For Each Loop

Enhanced for loop can be used to iterate through the elements of a collection or an array

Syntax:

```
for (datatype element: array/collection)
{
    statement(s);
}
```

Data type declared in the for each loop must match the data type of the array/collection that is iterated

Break & Continue Statements

- Break statement is used to terminate the loop and return the control to first statement after the loop.

  Syntax:

  break;

- Continue statement is used to skip the current iteration of the loop and jump to the next iteration of the loop. It can be used with any type of loop like for, while and do-while loop.

  Syntax:

  continue;

QASCRIPT

Methods

- Method is a block of code which performs a specific task
- Methods help developers organize their code
- Methods help us to define a task and reuse it throughout our code

Sample Task: Get Instructions for driving a car

- Step 1 – Insert the car keys
- Step 2 – Start the engine
- Step 3 – Press the accelerator

QS
QASCRIPT

Methods

Defining a Method:

- Use the task name “getInstructionsForDrivingACar” as the method name
- Tasks will form the different steps inside the method

Calling a Method:

- To use or call a method we can use the name of the method. Ex – getInstructionsForDrivingACar()

QS
QASCRIPT

Method Parameters

Consider a method which adds two numbers:

num1 = 5, num = 3

Step 1: result = num1 + num2

Step 2: Print result

This method will always print the same output i.e. 8

How can we add different numbers by using the same method?

We can do it by defining different inputs in the function

Method Parameters

Method with Parameters:

- num1, num2 inputs will be defined in the method definition
- Values of these inputs will be assigned when the method is called
- Same method can be used multiple times for different set of inputs

Examples:

add(2,3)

add(4,5)

Method Return Types

- Consider we want to multiply the output of add() method with another number
- How can we do it ?

  We can return the output in the method itself and then use it in another place
- This is done with the help of return types in methods

QASCRIPT

Built In Methods

- Java has already defined many methods in its library
- These methods are called built-in methods
- Built-in methods can be used by just calling them
- System.out.println() is a built in method
- Use dot operator to get access to many built in method