Summary for Written Test **Processing** 2020 v1.0

by THOMAS VAN KLINK

Creative Technology – M1: We create identity.

According to subjects mentioned in manual

Information taken from lectures, tutorials, reference and textbook

Introduction to Programming

Processing is a language for electronic arts, new media and visual design. Is a dialect of Java.

Code reads from top to bottom.

A Processing program is called a *sketch*.

Lines of code end with ;

Start your file with a

```
void setup() {
size(1000,1000)
}
```

to initialize your screen. No code before **size()** or **fullscreen()**

Basic functions:

background(), stroke(), fill(), noFill(), noStroke(), point(), line(), rect(), ellipse(), rectMode(), ellipseMode()

Basic layout:

```
void setup() {
size(1000,1000)
}
void draw() {
rect(500,500,50,50)
}
```

Runs only once, used to setup the canvas, create shared data structures.

Repeats as long as the program runs at the set framerate (default 60). Draws graphics and deals with interactions and animation.

When a piece of code is run: Translates code to Java > Compile into Java byte code > Execution as .jar

Variables, Types, Arithmetic

Common basic types:

- **byte** (small number) Can only store a small number, -128 to 127.
- **short** (larger number) Can store a larger number, -32,768 to 32,767.
- **int** (Integer) Can only contain whole numbers. Has a "limited" size.

- **float** (Floating point) Can be used for number with a floating point aka decimal point. Larger size.
- **boolean** (true/false) Can only be true or false, useful for control statements. Is not a string.
- **char** (character) Can store one character. Mostly used for keypresses.
- **string** (characters) Can store a sequence of characters. Mostly used for text.

Variables need both a type and name. Give ze variables meaningful names and avoid fixed Processing variables.

Write your variables name in **camelCase**, this is common practise.

Before you can use a variable you it is a good idea initialize it such as:

int Count = 50; | This is called an assignment statement

float Cookies = 37.5;

Aside from basic variables Processing has predefined variables which enable access to certain data at ease.

Some predefined types:

- **mouseX, mouseY** The position of the mouse/cursor on the screen.
- **pmouseX, pmouseY** The position of the mouse/cursor on the screen of the previous frame.
- **width, height** Width and height of the current sketch, not the size of the display of the computer.
- **keyCode** Numerical value of the last key that was pressed.
- **keyPressed** A function that is called every time a key is pressed. Value will be stored in **key**.
- **mousePressed** A function that is called every time the mouse button is pressed.
- **mouseButton** System variable which is set to the last of the three mouse buttons pressed.
- **frameCount** Total amount of frame drawn since the program has started.

Arithmetic's:

For most data types you can use the usual arithmetic operators:

- **float** +, -,*, / with the usual meaning
- **int**: +,-,* with the usual meaning.

Division (/) will be integer division. If both the numerator and divisor are integer. (11 / 4 = 2, not 2.75)

Another useful operator is modulo (%), computes the remainder of division.

For other data types these operators may or may not be defined so be careful with using them without checking the meaning of those operators for those certain data types.

Scopes:

Every pair of curly brackets (**{ }**) defines a local scope. Scopes can be nested.

Each variable that is declared exists from the point of declaration to the end of the scope.

You *cannot* use a variable outside of its scope. Declarations outside of any **{ }** are in Global Scope.

Variables in the global scope can be used everywhere (these are called global variables).

It's good practise to minimize the scope, this is easier to understand and is better for debugging.

If, If-Else, Booleans

If, if-else:

Within a usual method, all lines of code are executed after each other. Often you want to make these *conditional*.

You can decide to create two alternatives or even more.

Basic layouts:

```
if (condition) {
doThis…
}
```

The first example will run the code withing the brackets if the condition in the statement is met.

```
if (condition) {
doThis…
}
if (condition) {
doThis…
}
else {
doThat…
}
```

In the second examples you execute the first statement when the condition is met and then check for other condition is the alternatives.

```
if (condition) {
doThis…
}
else if (condition) {
doThis…
}
else if (condition){
doThat…
}
Else {
doSomething…
}
```

The third example on the right only moves on if the first condition is met and then only after the second and then only after the third etc.

Booleans:

Named after George Bool, pioneer of the study of logic. Invented an algebra based on only two values:

true and *false.*

Example of a boolean variable: **boolean isLegalAge= false;** Name is "isLegalAge", initialized to false.

When making complex decisions you often need to combine boolean values.

Boolean operators combine multiple values and expressions and combine them into a resultant boolean value.

Common boolean operators:

- AND (&&) – a AND b - ∧ Evaluates to true if both a and b are true.
- OR (||) - a OR b - ∨ Evaluates to true if A is true or B is true. At least one is true, both could be true.
- NOT (!) - !a - ¬ Evaluates to true only if A is false.

Relational operators (use in combination with Booleans)

- **==** Equality
- **!=** Inequality
- **<=** Less then or equal
- **<** Strictly less
- **>=** Greater then or equal
- **>** Strictly greater

Loops

While:

Basic layout:

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

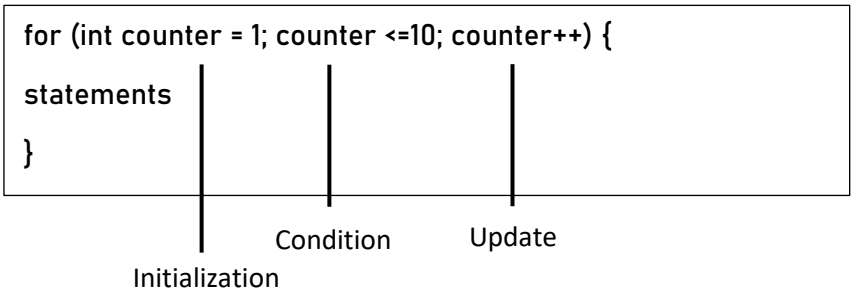
This is the body of the while

This loop will be repeated as long as the condition is true. If it's not the loop will stop.

Avoid infinite loops, they can make your program crash or freeze since they fill the memory quickly.

For's:

Basic layout:



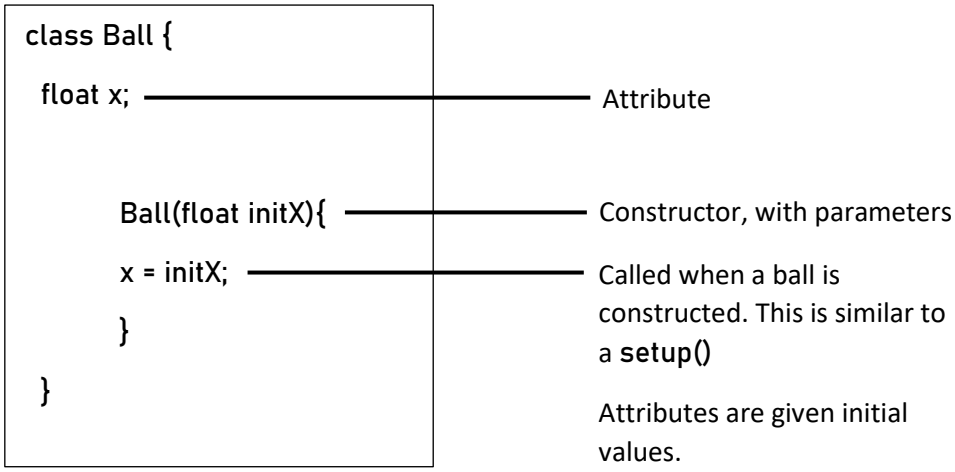
With this layout you can easily repeat certain statement an X number of times by changing the condition for the counter variable which is used only for counting the number of repeats in this case. You can use the counter to increase values in the statements per step as well.

The variable name **i** is also often used instead of counter.

Don't forget that the predefined method **draw()** is also a loop. Which uses frames as the counter.

Classes and Methods

Basic layout:



To add behaviour or functions:

Other variables introduced in this example should be added as attributes first.

```
void display(){
        fill(ballColor);
        stroke(0);
        ellipse(x,y,ballSize,ballSize);
        }
 }
```

Method, can contain any regular code.

After defining a class (on a new tab/file) you can call and use the class in the main tab/file.

Like so:

```
Ball ballA
void setup(){
        size(600,600);
        ballA = new Ball(100,100,20,color(255,0,0));
        }
void draw(){
        background(0);
        ballA.display();
}
```

Declare a variable of type/class Ball

Calls the constructor and passes parameters.

Is a object of type Ball

Calls the method **display()** of object ballA

To add another ball just add a new variable of type/call Ball: **Ball ballB,** and repeat the other code as well.

Methods:

Processing has multiple types of methods:

• Event methods, *update attributes in response to user events*	Change attributes	void
• Updates methods, *update attributes frame to frame*	Change attributes	void
• Display methods, *defines what graphical elements are used.*	Do not change attributes	void
• Queries, *returns a value that represents the state of the object*	Do not change attributes	non-void

It's good practise that every class manages its own attributes. This is called **encapsulation**.

You can/should use the different types of methods for that. For example, make a method that changes the colour of an element and then call it in the main class instead of changing a global variable and then passing that as a parameter.

For methods that only compute values you can use a non-void return type.

For example:

```
boolean mouseOverBall() {
boolean isOver = false;
if(pow(mouseX-x,2) + pow(mouseY-y,2) <=
pow(ballSize,2)) { isOver= true; }
return isOver;   }
```

In the example the keyword **return** is used to return a value, but it also means that that is the end of the method. Any code inside of the method after **return** will never be run of executed.

Methods can also use parameters, define them as: **void move(float tempMouseX, float tempMouseY){ }**

Use these temp variables as if they already contain the data which they will receive as parameter.

In the main tab it can look like this: **ballA.move(mouseX,mouseY);**

The variables that are passed there are called arguments, the action itself is a method call.

Methods can also be used in the main tab, they do not need to be from a class.

To summarize:

A class is a generic description of attributes and behaviour,

it bundles attributes and methods to update those attributes.

A object is an instant of a class with specific attributes and behaviour,

one for each relevant, interesting useful entity/concept.

Arrays

An array is a data structure, to store more of the same.

All elements of an array must be of the same *data type*. This means that object can be stored in array's too.

The elements are stored and accessed in sequence. **Counting starts at 0** then continues as usual.

To access the second element you use index 1, since you start from 0.

Basic layout:

```
int [] xpos = new int[50];
int [] ypos = new int [50];


for (int i = 0; i < xpos.length; i++)
{
        xpos[i] = 0;
        ypos[i] = 0;
}
```

Declares an array of type integer

You can also use **int xpos[];**

This creates a new array with 50 elements

A common type of for loop iterating through the elements of the array.

Starts at 0 (so index 1) and goes up to the length of the array.

Use method **length** on any array to obtain the current length.

The last legal element of a array is **length-1** for example: **xpos[xpos.length-1] = variable;**

You can also pass arrays as parameters.

An common array algorithm:

Dropping the first, moving all other one to the front, adding one to the end.

```
for (int i = 0; I < xpos.length-1; i++)
{
        xpos[i] = xpos[i+1];
        ypos[i] = ypos[i+1];
}
xpos[xpos.length-1] = mouseX;
ypos[ypos.length-1] = mouseY;
```

In this case using **xpos.length** would cause a major issue since you try to access a element which doesn't exit. This would make you program crash.

This is called a "buffer overflow" or "out-of-range" access. One of the most dangerous mistakes.

You are then accessing memory which doesn't belong to your program which may crash your program or give access to confidential data.

Keep in mind when to and not to use `-1` or just the length. Errors made with this ("off-by-one errors") are common and will make some sums incorrect.

You can also declare arrays of objects:

```
Planet[] planets = new Planet[9]

for (int I =0; i<planets.length; i++){
        planets[i] = new Planet(10*I, 20*I, 40);
}
```

Planet[] means it's an array of type **Planet**

Creates a new array with 9 planets.

Common for loop to iterate through the array of objects.

Parameters are changed each iteration

General Information

Primitive types vs Data structures:

Primitive types are types such as **int, float, char**. Variables of those types always contain a value of that type. They behave mostly like variables from mathematics.

Data structures are classes, but also arrays, lists etc define data structures. A variable of those types contains a reference to a data structure, not the data itself. Before you can use these, you must create a data structure and assign it to them.

See them as just counting money in your hand vs using a bank account.

Matrixes:

Use push and pop matrix to move the origin to a more convenient place.

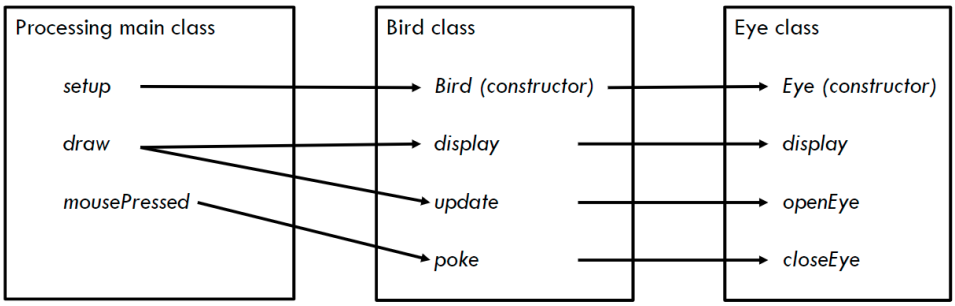
For example:

```
pushMatrix();          Stores current settings and pushes everything to a matrix
translate(x,y);        Moves origin of coordinate system to current position
ellipse(0,0,10,10)     Use the new origin for drawing shapes
...
popMatrix();           Restores saved settings and pops the matrix
```

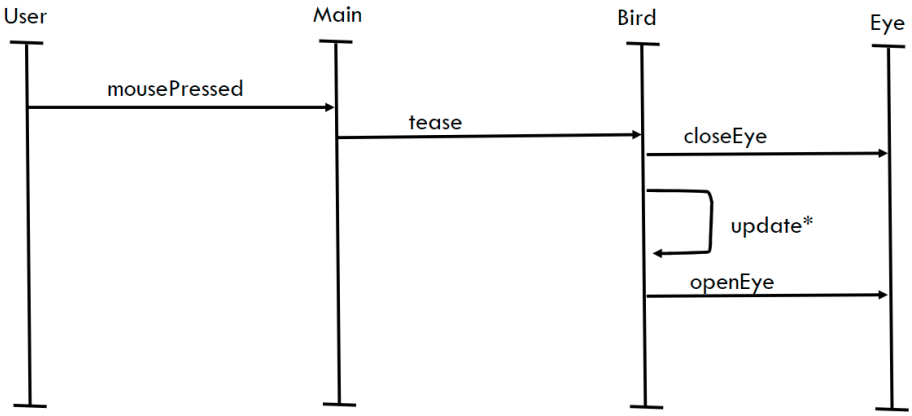
Flowcharts and cell graphs:

You should be able to understand a flow charts of you programs flow.

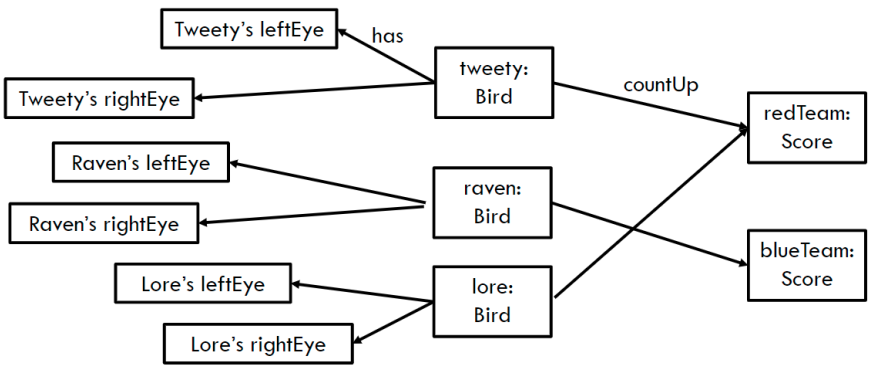
You should be able to understand a call graph of your method calls:



You should be able to understand a sequence diagram of a sequence of events:



You should be able to understand an object diagram:



Common structure of Processing program:

Main Processing Tab	Class A	Class B
`global variables` Use for information concerning the entire update, and variables for important objects.	`attributes` What's important to know about an object of this class	`attributes` What's important to know about an object of this class
`setup` Initialize the global variables, set size, create global objects	`constructor` Initializes the attributes for a new object	`constructor` Initializes the attributes for a new object
`draw` All code that drives the display of objects, All code that drives the update	`display methods` One or more methods to display the different parts	`display methods` One or more methods to display the different parts
`event handlers` This are predefined methods such as mousePressed or keyPressed	`update methods` These methods will update attributes as time (frames) go by.	`update methods` These methods will update attributes as time (frames) go by.
`utility methods` Useful methods that don't belong to any particular class.	`event handlers` Update attributes in response to events.	`event handlers` Update attributes in response to events.

Numerical accuracy:

Avoid comparing floats with == since these calculations are sometimes rounded in a weird way which causes issues with the calculations and accuracy.