

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ЗАПОРІЗЬКА ПОЛІТЕХНІКА»

Кафедра ПЗ

МЕТОДИЧНІ ВКАЗІВКИ

до самостійної роботи з дисципліни
«Інженерія програмного забезпечення вбудованих систем»
для студентів ОР «бакалавр»
за спеціальністю 121 «Інженерія програмного забезпечення»

Методичні вказівки до самостійної роботи з дисципліни «Інженерія програмного забезпечення вбудованих систем» для студентів ОР «бакалавр» за спеціальністю 121 «Інженерія програмного забезпечення» /Укл.: А.В. Пархоменко, О.М. Гладкова – Запоріжжя: НУ «Запорізька політехніка», 2023. – 39 с.

Укладачі: А.В. Пархоменко, к.т.н., доцент кафедри ПЗ,
О.М. Гладкова, к.т.н., доцент кафедри ПЗ.

Рецензент: І.Я. Зеленьова, к.т.н., доц. кафедри КСМ.

Відповідальний
за випуск: С.О. Субботін, д.т.н., професор, зав. кафедри ПЗ.

Затверджено
на засіданні кафедри
Програмних засобів
Протокол № 12
від 09.06.2023 р.

ЗМІСТ

1 Самостійна робота 1 Основи роботи з Processing	5
1.1 Теоретичні відомості	5
1.1.1 Інсталяція Processing.....	5
1.1.2 Робота з головним вікном програми	5
1.1.3 Мова програмування Processing	7
1.1.4 Основні функції.....	7
1.1.5 Приклади програм.....	8
1.2 Завдання до виконання самостійної роботи.....	11
1.3 Контрольні запитання	12
1.4 Зміст звіту.....	13
2 Самостійна робота 2 Робота зі змінними у Processing	14
2.1 Теоретичні відомості	14
2.1.1 Змінні створені користувачем	14
2.1.2 Змінні за замовчанням	15
2.1.3 Використання математичних операцій та операторів	16
2.1.4 Використання циклу for	17
2.1.5 Використання функції Random	18
2.2 Завдання до виконання самостійної роботи.....	19
2.3 Контрольні запитання	20
2.4 Зміст звіту.....	20
3 Самостійна робота 3 Взаємодія Processing з користувачем	
.....	21
3.1 Теоретичні відомості	21
3.1.1 Використання <i>mousePressed</i> та <i>keyPressed</i> як змінних	22

3.1.2 Використання функції <i>if</i> зі значеннями <i>mouseX</i> та <i>mouseY</i>	24
3.2 Завдання до виконання самостійної роботи.....	25
3.3 Контрольні запитання	26
3.4 Зміст звіту.....	26
4 Самостійна робота 4 Послідовний інтерфейс вводу/виводу	27
4.1 Теоретичні відомості	27
4.2 Завдання до виконання самостійної роботи.....	32
4.3 Контрольні запитання	32
4.4 Зміст звіту	32
5 Самостійна робота 5 Взаємодія Processing та Arduino	33
5.1 Теоретичні відомості	33
5.2 Завдання до виконання самостійної роботи.....	37
5.3 Контрольні питання.....	37
5.4 Зміст звіту	37
6 Самостійна робота 6 Розробка графічного інтерфейсу з використанням бібліотеки Firmata	38
6.1 Теоретичні відомості	38
6.2 Завдання до виконання самостійної роботи.....	38
6.3 Контрольні питання.....	38
6.4 Зміст звіту.....	39
ДЖЕРЕЛА ІНФОРМАЦІЇ	39

1 САМОСТІЙНА РОБОТА 1

ОСНОВИ РОБОТИ З PROCESSING

Мета роботи: у рамках цієї роботи необхідно ознайомитись з базовими поняттями Processing, вивчити основні функції для побудови графічних елементів і роботи з ними.

1.1 Теоретичні відомості

1.1.1 Інсталяція Processing

Processing – це IDE призначене для вивчення основ інтерактивної комп’ютерної графіки (створення анімації, інтерфейсів та простих ігор).

Processing є портативним, тобто не потребує встановлення. Воно є безкоштовним і знаходиться у вільному доступі. Необхідно завантажити архів програми (посилання на завантаження знаходиться нижче).

У цій роботі буде використовуватися «Processing 3.3.1». Завантажити архів програми можна за посиланням: <https://processing.org/download/?processing>.

Далі необхідно видобути папку з архіву, вміст папки виглядає як зображено на рис 1.1.

1.1.2 Робота з головним вікном програми

Якщо відкрити файл processing.exe то ви побачите вікно зображене на рис. 1.2. Це так зване вікно створення ескізу (sketch), тобто вихідного тексту програми. Воно складається з наступних елементів: меню, панель інструментів, вкладки, текстовий редактор, вікно повідомлень та консоль.

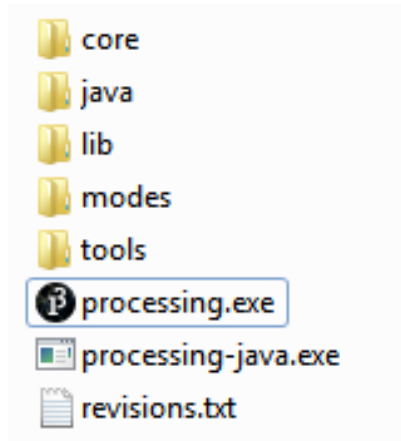


Рисунок 1.1 – Вміст папки після видобутку з архіву

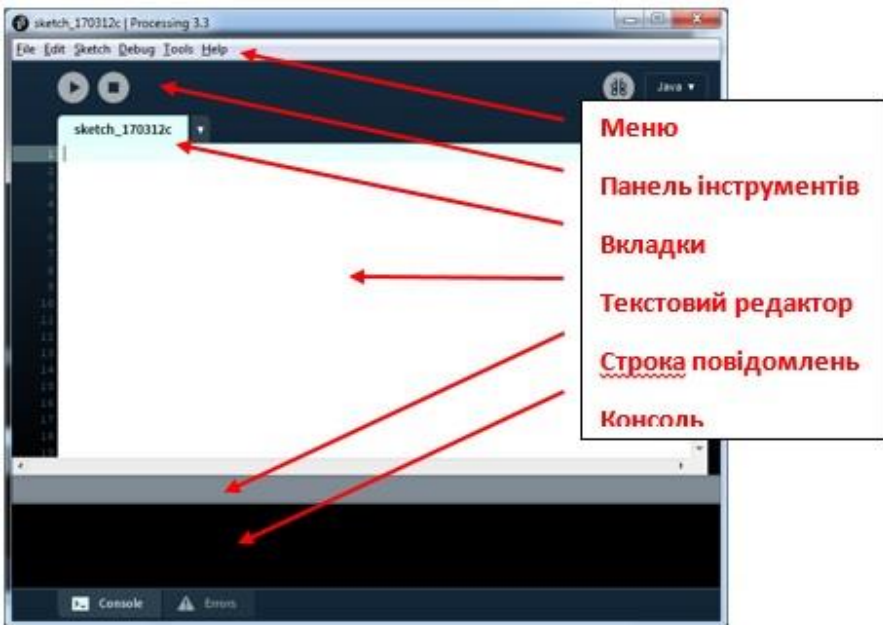


Рисунок 1.2 – Вікно створення ескізу Processing

1.1.3 Мова програмування Processing

Мова програмування Processing є діалектом Java але з деякими особливостями для роботи з графікою та інтерактивним наповненням (рис.1.3). Processing багато чого взяв з PostScript (що послужило основою для формату PDF) та OpenGL (специфікація 3D графіки).

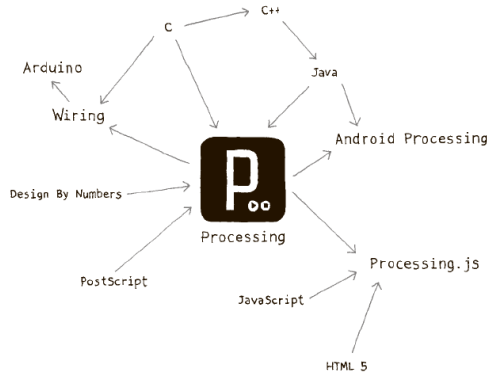


Рисунок 1.3 – Взаємозв’язок Processing та інших мов програмування

1.1.4 Основні функції

Щоб ознайомитись з можливостями та побудувати першу програму треба оволодіти наступними базовими функціями перерахованими нижче.

`void setup()` та `void draw()` // основні функції роботи програми, усі інші функції пишуться у середині цих двох циклів. `void setup()` виконується один раз на початку роботи програми. `void draw()` – це функція нескінченного циклу, вона виконується постійно.

`size(X, Y);` // створює робочу область (або вікно) розміром X на Y пікселів, за замовчанням вона 100 на 100 пікселів.

`background(a);` // задає колір фону. `background(a)` еквівалентно запису `background(a,a,a)`, де три змінні відповідають основним кольорам кольірної моделі RGB: червоному, зеленому й синьому.

`stroke(R,G,B);` // задає колір лінії фігури (еліпсу, прямокутника), якщо його не вказати за замовчанням контур фігури буде чорним.

`noStroke();` // фігура малюється без контуру.

`strokeWeight(p);` // задає товщину лінії у пікселях.

`fill(R, G, B);` // вказує колір заповнення фігури, можна додати четвертий параметр до цієї функції `T` - це прозорість, за замовчанням, якщо не вказувати цей параметр, то він дорівнює 255, тобто 100% не прозоре.

`fill(A);` // задає колір заповнення еліпсу як чорний, білий і відтінки сірого, відповідає наступному RGB запису `fill(A, A, A)`; наприклад `fill(128)` відповідає `fill(128, 128, 128)` – середній сірий.

`noFill();` // вимикає заповнення фігури, тобто малюється тільки контур.

Форми:

`point(x,y);` // малює точку.

`line(x1, y1, x2, y2);` // малює лінію з точки (x_1, y_1) до точки (x_2, y_2) .

`triangle(x1, y1, x2, y2, x3, y3);` // малює трикутник за трьома точкам.

`ellipse(x, y, w, h);` // вказує положення еліпсу на екрані та його розміри, де x та y це координати центру еліпсу, w – діаметр по горизонталі, h – діаметр по вертикалі.

`arc(x, y, w, h, start, stop);` // малює частину еліпсу, п'ятий та шостий параметр вказує кут початку дуги та кут закінчення дуги відповідно, параметри кута задаються як радіан використовуючи значення `PI(3,14159)`. Змінні які можна використовувати у якості параметрів кута початку і зупину: `0`, `PI`, `QUARTER_PI`, `HALF_PI`, and `TWO_PI`. Де `PI` – 180 градусів.

`rect(x, y, w, h);` // малює прямокутник, починаючи з лівої верхньої точки (x, y) , за параметрами w – довжина, h – висота. Можна також додати п'ятий параметр t – радіус округлення кутів прямокутника.

`guat(x1, y1, x2, y2, x3, y3, x4, y4);` // малює довільний чотирикутник.

1.1.5 Приклади програм

Відкрийте середовище Processing, скопіюйте програму, яка наведена нижче у прикладі, збережіть програму на диску (File->Save)

та клікніть на панелі інструментів кнопку **Run (Запуск)**, це запустить компіляцію програми.

Приклад №1:

```
void setup() {  
    size(640, 480);  
}  
void draw() {  
    background(0);  
}
```

Результатом роботи цієї короткої програми повинно бути нове вікно розміром 640 на 480 пікселів, зафарбоване у чорний колір.

Приклад №2:

```
void setup() {  
    size(180, 320);  
}  
void draw() {  
    background(255,0,0);  
}
```

Результатом роботи цієї короткої програми повинно бути нове вікно розміром 180 на 320 пікселів, зафарбоване у червоний колір.

Приклад №3:

```
void setup() {  
    size(200, 200);  
}  
void draw() {  
    background(0);  
    // побудова еліпсу  
    stroke(255); //білий контур еліпсу  
    fill(255, 0, 0); // червоне заповнення еліпсу  
    ellipse(100, 100, 150, 75); // параметри еліпсу  
}
```

Приклад №4:

```

void setup() {
    size(200, 200);
}

void draw() {
    background(0);
    // еліпс
    stroke(255);
    fill(255, 0, 0);
    ellipse(100, 100, 150, 75);
    //прямокутник
    stroke(255, 255, 0);
    fill(0, 0, 255, 100);
    rect(50, 50, 120, 60);
}

```

Як можна бачити з програми, колір елемента задається перед тим як об'явити сам елемент та його параметри.

Крім базових форм користувач може побудувати довільну фігуру як послідовність з'єднаних точок. Для цього використовуються наступні функції:

`beginShape();` // функція яка свідчить про початок довільної фігури

`vertex();` // функція яка задає кожну пару x та y координат фігури

`endShape();` // завершення побудови фігури

Розглянемо приклад створення довільної фігури. Результат роботи зображено на рисунку 1.4.

Приклад №5:

```

size(450, 120);
beginShape();
vertex(150, 110);
vertex(300, 40);
vertex(300, 110);
vertex(150, 110);
vertex(150, 40);
vertex(300, 40);

```

```
vertex(225, 10);  
vertex(150, 40);  
vertex(300, 110);  
endShape(CLOSE);
```

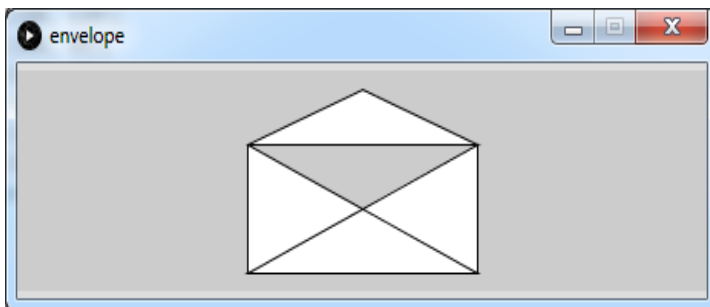


Рисунок 1.4 – Приклад створення довільної фігури

1.2 Завдання до виконання самостійної роботи

1.2.1 Ознайомитися з теоретичними відомостями.

1.2.2 Запустити графічний редактор IDE Processing.

1.2.3 Створити робоче вікно розмірами 400 на 400 пікселів.

Намалювати одну з фігур зображених на рисунку 1.5.

1.2.4 Написати програму, яка створює 3 кола червоного, зеленого й синього кольору. Прозорість кожного кола на 75 одиниць менша за попереднє, починаючи з повністю непрозорого. Контур на 10 одиниць більше починаючи з нуля (рис 1.6).

1.2.5 Оформити звіт до самостійної роботи.

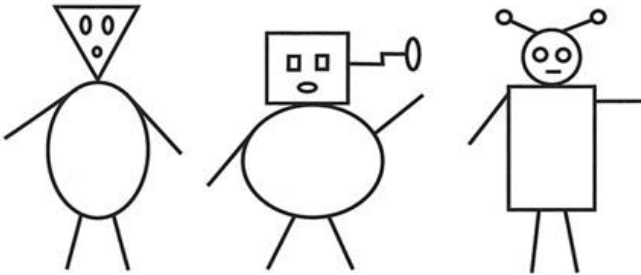


Рисунок 1.5 – Фігури

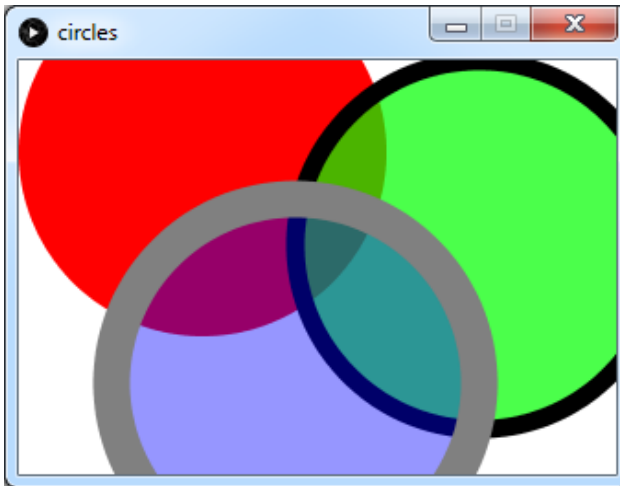


Рисунок 1.6 – Зображення кіл

1.3 Контрольні запитання

1.3.1 Які параметри робочого вікна встановлені за замовчанням?

1.3.2 Якими будуть значення функції заповнення для чорного, білого та середнього сірого кольору?

1.3.3 Який з нижче наведених еліпсів буде найпрозорішим?

- `fill(255, 0, 0, 90);`
`ellipse(100, 100, 150, 75);`

- `fill(0, 0, 0, 100);`
`ellipse(50, 50, 120, 60);`
- `fill(0, 100, 0, 200);`
`ellipse(50, 80, 60, 60);`

1.3.4 Вкажіть довжину у сантиметрах лінії:

```
line(0, 50, 50, 50);
```

1.4 Зміст звіту

1.4.1 Мета самостійної роботи.

1.4.2 Короткі теоретичні відомості.

1.4.3 Код програми.

1.4.4 Знімки екрану, які відображають виконані завдання.

1.4.5 Висновки до роботи.

2 САМОСТІЙНА РОБОТА 2

РОБОТА ЗІ ЗМІННИМИ У PROCESSING

Мета роботи: ознайомитись з поняттям змінної, які змінні існують у середовищі Processing за замовчанням. Навчитися створювати власні змінні та застосовувати їх при написанні програм.

2.1 Теоретичні відомості

Змінна – це значення, яке зберігається у пам'яті під своїм власним ім'ям.

Якщо при написанні програми якесь значення використовується декілька разів, то необхідно для зручності ініціалізувати змінну, тобто визначити та оголосити її.

2.1.1 Змінні створені користувачем

Розглянемо приклад програми:

Приклад №1:

```
size (480,360);  
int diam = 100; // визначаємо що є значення 100 та  
оголошуємо його як цілочислову змінну (int) з ім'ям  
diam  
ellipse (120, 120, diam, diam); // в якості  
діаметрів еліпсу використовується змінна diam, тобто  
значення діаметрів = 100  
ellipse (300, 300, diam, diam);
```

У цьому прикладі було ініціалізовано змінну `int diam=100`. У програмі вона використовувалась для встановлення діаметру еліпсів (функція `ellipse()`). Тобто у результаті роботи програми ми отримали 2 однакових еліпси, які розміщені у різних місцях вікна відображення. Далі, для того, щоб змінити значення діаметрів нам не потрібно у чотирьох місцях в програмі змінювати значення діаметрів, а потрібно тільки змінити значення змінної `diam`. Це робить програму простішою до змін.

Ці змінні можна назвати змінними користувача. Ім'я змінної повинно бути зрозумілим, не повинно містити пропуски, числа на початку ім'я, символу #, та не повинно співпадати з ключовими словами програми, наприклад `void`.

Окрім тих змінних, які оголосив користувач, середовище Processing також має так звані змінні за замовчанням, які перелічені нижче.

2.1.2 Змінні за замовчанням

`mouseX, mouseY` // використовує положення курсора в робочому екрані по осі X та Y

`pmouseX, pmouseY` // системна змінна, містить положення миші у кадрі, що передує поточному кадру

`height, width` // відповідають значенням висоти и ширини робочого екрану, які задаються функцією `size()`

Код прикладів використання змінних за замовчанням наведені на рисунках 2.1-2.2.

```
void setup() {
    size(480, 120);
}
void draw() {
    smooth();
    line(0, 0, width, height); //рисує лінію від(0,0) до(480,120)
    line(width, 0, 0, height); //рисує лінію від(480,0) до(0,120)
    ellipse(width/2, height/2, 60, 60);
}
```

Рисунок 2.1 – Приклад №2

Щоб зрозуміти як працює цей приклад, змініть значення у функції `size()` на якесь інше (наприклад, `size(280, 320);`).

Розглянемо інший приклад (рис 2.2). В цьому прикладі параметрами еліпсу є: центр еліпсу задається положенням курсора, а діаметр еліпсу по ширині рівний висоті робочого вікна, яке вказано у функції `size()`, тобто 150.

```

void setup () {
    size (400,150);
}
void draw () {
    background (0);
    //побудова еліпсу
    stroke (255); //білий контур еліпсу
    fill (255,0,0); //червоне заповнення еліпсу
    ellipse (mouseX, mouseY, height, 100); //параметри еліпсу
}

```

Рисунок 2.2 – Приклад №3

2.1.3 Використання математичних операції та операторів

Бачити математичні залежності та використовувати математику при написанні програми є також важливим вмінням. Символи «+», «-», «*», «/», «=» - називаються операторами, які використовуються для створення основних арифметичних операцій. Коли вони знаходяться між двома значеннями то створюють вираз. Далі наведено декілька прикладів коду, де математичні операції використовуються для задання положення чотирикутника на екрані (див. рис. 2.3 – 2.4).

```

void setup () {
    size (400,150);
}
void draw () {
    background (0);
    //побудова прямокутника
    stroke (255); //білий контур прямокутника
    fill (255,0,0); //червоне заповнення прямокутника
    rect (mouseX, height-mouseY, 100, 100); //параметри прямокутника
}

```

Рисунок 2.3 – Приклад №4


```

int x = 25;
int h = 20;
int y = 25;
void setup () {
    size (480, 120);
}
void draw () {
    rect (x, y, 300, h); //верхній ряд прямокутників
    x = x + 100;
    rect (x, y + h, 300, h); //середній ряд прямокутників
    x = x - 250;
    rect (x, y + h*2, 300, h); //нижній ряд прямокутників
}

```

Рисунок 2.4 – Приклад №5

Деякі вирази використовуються як скорочення:

```

x += 15; // те ж саме, що x = x + 15
x++; // те ж саме, що x = x + 1

```

2.1.4 Використання циклу for

У цій самостійній роботі слід також згадати про цикл for. Його доречно використовувати коли у програмі команди часто повторюють одна одну з незначними змінами. Розглянемо декілька прикладів коду (рис 2.5 – 2.6).

У цих двох прикладах представлена одна й та сама програма, але у першому прикладі без використання циклу for, а у другому реалізована з циклом for.

```

void setup(){
    size(480, 120);
}
void draw(){
    smooth();
    strokeWeight(8);
    line(20, 40, 80, 80);
    line(80, 40, 140, 80);
    line(140, 40, 200, 80);
    line(200, 40, 260, 80);
    line(260, 40, 320, 80);
    line(320, 40, 380, 80);
    line(380, 40, 440, 80);
}

```

Рисунок 2.5 – Приклад №6

```

void setup(){
    size(480, 120);
}
void draw(){
    smooth();
    strokeWeight(8);
    for (int i = 20; i < 400; i += 60) {
        line(i, 40, i + 60, 80);
    }
}

```

Рисунок 2.6 – Приклад №7

2.1.5 Використання функції Random

Далі розглянемо приклади з функцією random() (рис 2.7 – 2.8). Вони демонструють різницю між роботою функцій void setup() та void draw().

random(x); // функція, яка в кожному циклі роботи програми генерує довільне значення від 0 до x.

```
float circleX, circleY;

void setup(){
  size(280, 360);
}

void draw(){
  background(0);
  circleX=random(width);
  circleY=random(height);
  ellipse(circleX, circleY, 50, 50);
}
```

Рисунок 2.7 – Приклад №8

```
float circleX, circleY;

void setup(){
  size(280, 360);
  background(0);
}

void draw(){
  circleX=random(width);
  circleY=random(height);
  ellipse(circleX, circleY, 50, 50);
}
```

Рисунок 2.8 – Приклад №9

2.2 Завдання до виконання самостійної роботи

2.2.1 Ознайомитися з теоретичними відомостями.

2.2.2 Запустити графічний редактор IDE Processing.

2.2.3 Розробити програму, яка відображає коло, яке переміщується з лівої сторони робочого вікна Processing до правої, і коли досягає крайнього правого положення знову починає рухатись зліва на право.

2.2.4 Розробити програму, яка створює робоче вікно світло-блакитного кольору з двома прямокутниками по центру, один з яких поступово змінює колір з білого до чорного, а інший з чорного на білий і навпаки.

2.2.5 Розробити програму, яка буде малювати нескінченну лінію за курсором мишки.

2.2.6 Розробити програму, яка відображає коло, яке циклічно переміщується з крайньої лівої сторони робочого вікна Processing до крайньої правої сторони і в зворотному напрямку.

2.2.7 Оформити звіт до самостійної роботи.

2.3 Контрольні запитання

2.3.1 Що таке змінна? Які бувають змінні?

2.3.2 Як ініціалізувати змінну? Які символи не повинні міститись у назві змінної?

2.3.3 Опишіть різницю між роботою функцій `void setup()` та `void draw()`?

2.3.4 Як ви зрозуміли роботу змінних `pmouseX` та `pmouseY`?

2.4 Зміст звіту

2.4.1 Мета самостійної роботи.

2.4.2 Короткі теоретичні відомості.

2.4.3 Код програми.

2.4.4 Знімки екрану, які відображають виконані завдання.

2.4.5 Висновки до роботи.

3 САМОСТІЙНА РОБОТА 3

ВЗАЄМОДІЯ PROCESSING З КОРИСТУВАЧЕМ

Мета роботи: навчитися програмно реалізовувати взаємодію користувача з Processing за допомогою таких подій як: клік миші та натиснення клавіші на клавіатурі.

3.1 Теоретичні відомості

На прикладі наступної програми розглянемо як обробляються натискання клавіші миші або клавіші клавіатури. Напишіть у Processing приклад наведений нижче (рис. 3.1), розберіть роботу програми уважно.

У цьому прикладі оголошено дві функції `mousePressed()` та `keyPressed()`. Як можна зрозуміти з назв, ці дві функції переривають роботу програми та викликаються коли натиснута кнопка миші або клавіатури.

```
void setup() {  
}  
void draw() {  
  
void mousePressed() {  
    background(255, 0, 0);  
}  
void keyPressed() {  
    background(0, 0, 255);  
}
```

Рисунок 3.1 – Приклад №1

Розберіть наступну програму (рис. 3.2) для того щоб зрозуміти як працює функція `mousePressed()`.

```

float x = 50;
boolean go = false;
void setup() {
    size(300, 150);
}
void draw() {
    background(0);
    fill(255);
    ellipse(x, 75, 20, 20);
    if (go) {
        x = x + 1;
    }
}

void mousePressed() {
    go = true;
}

```

Рисунок 3.2 – Приклад №2

3.1.1 Використання mousePressed та keyPressed як змінних

MousePressed та keyPressed можна використовувати як змінну. Ця змінна має тип даних Boolean, тобто True або False. Коли клавіша або кнопка натиснута, вона дорівнює True. Розберіть наступні приклади і ви зрозумієте як використовувати змінні mousePressed та keyPressed.

Перший приклад ілюструє роботу (рис. 3.3) mousePressed як змінної для реалізації руху кола, як і в попередньому прикладі. Але результат роботи програми дещо відмінний.

У другому прикладі (рис 3.4) код в середині функції if спрацьовує тоді, коли кнопка миші натиснута. Коли вона не натиснута – цей фрагмент коду ігнорується.

У цих двох прикладах також ілюструється альтернативи запису функції. В обох випадках вона перевіряє чи натиснута кнопка миші, тобто чи є це значення логічною 1 (true).

Програма може містити набагато більше структур if та else. Декілька if можуть бути побудовані в довгу серію або вбудовані одна в одну, як показано в наступному прикладі. В цьому ж прикладі

зверніть увагу на те, як можна перевірити, яка саме кнопка миші натиснута (рис. 3.5).

```
float x = 50;
void setup() {
    size(300, 150);
}
void draw() {
    background(0);
    fill(255);
    ellipse(x, 75, 20, 20);
    if (mousePressed) {
        x = x + 1;
    }
}
```

Рисунок 3.3 – Приклад №3

```
void setup() {
    size(240, 120);
    smooth();
    strokeWeight(30);
}
void draw() {
    background(255);
    stroke(102);
    if (mousePressed == true) {
        stroke(0);
    }
    line(0, 70, width, 70);
}
```

Рисунок 3.4 – Приклад №4

```

int d = 50;

void setup() {
  size(200, 200);
}
void draw() {
  background(200);
  if (mousePressed) {
    if (mouseButton == LEFT) {
      fill(255);
    } else {
      fill(0);
    }
    ellipse(height/2, width/2, d, d);
  }
}

```

Рисунок 3.5 – Приклад №5

3.1.2 Використання функції if зі значеннями mouseX та mouseY

Функція if може бути використана зі значеннями mouseX та mouseY для перевірки положення курсору на екрані (рис 3.6). Для написання програми, яка має такі графічні інтерфейси як кнопки, смуга прокрутки, ми повинні знати коли курсор миші буде знаходитись над ними.

Для створення умовних виразів використовуються оператори відношення: '==', '!=', '>', '<', '>=', '<='. А також логічні оператори: AND (&&), OR (||), NOT (!).


```

void setup(){
    size(300, 150);
}
void draw(){
    background(0);
    if((mouseX > 200) || (mouseX < 100)){
        background(200, 0, 200);
    }
    stroke(200);
    strokeWeight(5);
    line(100, 0 , 100, width);
    line(200, 0 , 200, width);
}

```

Рисунок 3.6 – Приклад №6

3.2 Завдання до виконання самостійної роботи

3.2.1 Ознайомитися з теоретичними відомостями.

3.2.2 Виконати завдання згідно з варіантом студента за журналом:

Варіант 1

1) Створити робоче вікно розмірами 640 на 480 пікселів чорного кольору. Розділити цю область двома лініями білого кольору на чотири частини. Коли курсор миші переміщується над якоюсь частиною то вона змінює колір з чорного на будь-який інший.

2) Створити робоче вікно розмірами 480 на 120 пікселів та біле коло розмірами 50 на 50, яке рухається за курсором миші. Коли клавіша миші натиснута, колір кола змінюється на чорний.

3) Модифікуйте програму з колом, яке рухається. Зробіть так, що коли клавіша миші натиснута в перший раз, коло починає рухатись, коли клавіша миші натиснута вдруге, коло зупиняється.

Варіант 2

1) Створити робоче вікно розмірами 640 на 480 пікселів чорного кольору. Розділити цю область двома лініями білого кольору на чотири частини. Послідовно по годинниковій стрілці змінювати колір однієї частини вікна з чорного на будь-який інший.

2) Написати програму переміщення кола з однієї сторони вікна до іншої, і коли коло перетинає половину вікна, то коло та фон змінюють колір.

3) Модифікуйте програму з колом, яке рухається. Зробіть так, щоб після натискання лівої клавіші мишки коло починало рухатись, після натискання правої – коло зупинялося.

3.2.3 Оформити звіт до самостійної роботи.

3.3 Контрольні запитання

3.3.1 MousePressed як функція та як змінна, принципи роботи?

3.3.2 KeyPressed як функція та як змінна, принципи роботи?

3.3.3 В чому помилка у наступній програмі (рис. 3.7)? Чому не відображається коло на екрані?

```
float circleX = 0;

void setup() {
    size(640, 360);
}
void draw() {
    background(50);
    fill(255);
    ellipse(circleX, 180, 24, 24);
    circleX = circleX + random(-2, 0);
}
```

Рисунок 3.7 – Код з помилкою

3.4 Зміст звіту

3.4.1 Мета самостійної роботи.

3.4.2 Короткі теоретичні відомості.

3.4.3 Код програми.

3.4.4 Знімки екрану, які відображають виконані завдання.

3.4.5 Висновки до роботи.

4 САМОСТІЙНА РОБОТА 4

ПОСЛІДОВНИЙ ІНТЕРФЕЙС ВВОДУ/ВИВОДУ

Мета роботи: на простих прикладах освоїти принципи роботи з монітором послідовного порту середовищ розробки Processing та Arduino.

4.1 Теоретичні відомості

У середовищі Processing та Arduino існують монітори послідовного порту для вводу та виводу інформації, за допомогою яких користувач може «спілкуватися» з пристроєм підключеним до послідовного порту комп'ютера/ноутбука. У Processing відображення інформації з монітору послідовного порту можна виводити знизу вікна середовища розробки. В Arduino монітор послідовного порту викликається за допомогою клавіші «Serial Monitor», яка знаходиться на панелі інструментів (рис.4.1).

Розглянемо роботу монітору послідовного порту на прикладі Arduino.

Arduino має вбудований контролер для послідовної передачі даних, який може використовуватись як для зв'язку між Arduino пристроями, так і для зв'язку з комп'ютером. В Arduino існує спеціальна функція для роботи з цим контролером – Serial.

Serial посилає один байт за один раз. Для Arduino, байти – це тип даних, який може зберігати значення від 0 до 255; він працює як цілочисловий формат даних int, але з набагато меншим діапазоном. Великі номери відправляються розбиваючи їх в список байтів, а після відбувається їх повторна збірка.

Далі розглянемо детально функції для роботи з послідовною передачею та читанням даних, та приклад їх використання у програмі.

`Serial.begin(швидкість_передачі);`

Встановлює швидкість передачі інформації СОМ порту в «біт в секунду» для послідовної передачі даних. Для того щоб підтримувати зв'язок з комп'ютером, використовується одна з нормованих швидкостей: 300, 1200, 2400, 4800, 9600, 19200, 38400, 57600, або 115200 (наприклад, `Serial.begin(4800);`). За замовчанням в моніторі послідовного порту Arduino встановлена швидкість 9600 біт/сек.

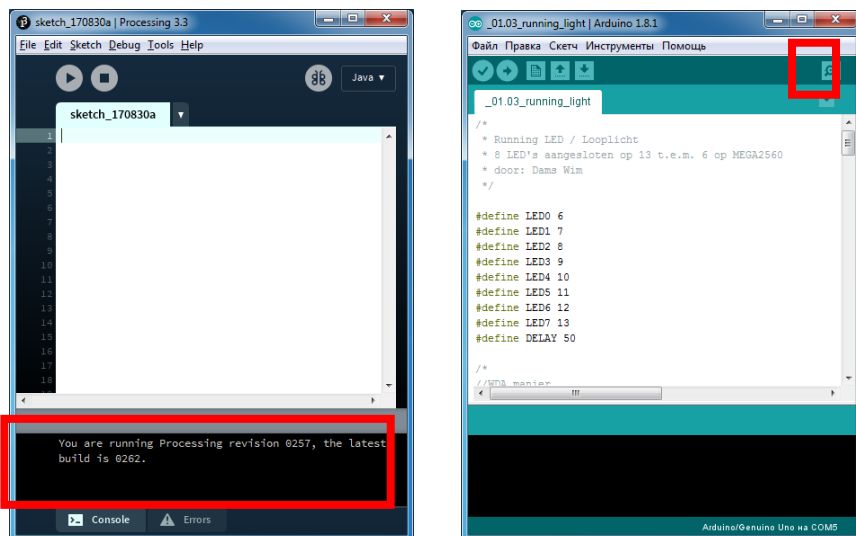


Рисунок 4.1 – Монітор послідовного порту Processing та Arduino

```
Serial.available();
```

Байти які надходять з послідовного порту потрапляють до буферу мікроконтролера, звідки програма може їх зчитати. Функція повертає кількість накопичених в буфері байт. Послідовний буфер може зберігати до 128 байт.

Повертає значення типу `uint8_t` (`typedef uint8_t byte;`) – кількість байт, які знаходяться у доступі для читання у послідовному буфері, або 0, якщо нічого не доступне.

```
Serial.read();
```

Зчитує наступний байт з буферу послідовного порту. Повертає перший доступний байт вхідних даних з послідовного порту, або -1 якщо нема вхідних даних.

```
incomingByte = Serial.read(); // зчитує байт
Serial.write(uint8_t c)
```

Виклик:

```
Serial.write(val);
Serial.write(str);
Serial.write(buf, len);
```

Записує данні у послідовний порт. Данні відправляються як байт або послідовність байтів; для відправки символної інформації слід використовувати функцію `print()`.

Параметри:

`val`: змінна для передачі, як одиничний байт

`str`: рядок для передачі, як послідовність байт

`buf`: масив для передачі, як послідовність байт

`len`: довжина масиву

```
Serial.flush();
```

Очищує вхідний буфер послідовного порту. Данні, які знаходяться у буфері втрачаються, і подальший виклик `Serial.read()` або `Serial.available()` будуть мати сенс для даних, які отримані після виклику `Serial.flush()`.

```
Serial.print()
```

Вивід даних до послідовного порту.

Функції `print` спадкується класом `HardwareSerial` від класу `Print` (`\hardware\cores\arduino\Print.h`)

Функція має декілька форм виклику в залежності від типу та формату даних, що виводяться:

`Serial.print(b, DEC);` виводить ASCII-рядок – десяткове представлення числа `b`.

```
int b = 79;
Serial.print(b, DEC); //виведе рядок «79»
```

`Serial.print(b, HEX);` шістнадцяткове представлення.

```
int b = 79;
Serial.print(b, HEX); //виведе «4F»
```

`Serial.print(b, OCT);` вісімкове подання числа `b`.

```
int b = 79;
Serial.print(b, OCT); //виведе «117»
```

`Serial.print(b, BIN);` двійкове подання числа `b`.

```
int b = 79;
Serial.print(b, BIN); //виведе «1001111»
```

`Serial.print(str)` якщо `str` – рядок або масив символів, побайтно передає `str` до COM-порту.

```
char bytes[3] = {79, 80, 81}; //масив з 3 байт зі
значеннями 79,80,81
Serial.print("Here our bytes:"); //виведе рядок
«Here our bytes:»
Serial.print(bytes); //виведе 3 символи з кодами
79,80,81
//це символи «OPQ»
```

`Serial.print(b)` якщо `b` має тип `byte` або `char`, виведе до порту саме число `b`.


```
char b = 79;
Serial.print(b); //виведе до порту символ «O»
```

`Serial.print(b)` якщо `b` має цілий тип, виведе до порту десяткове значення числа `b`.

```
int b = 79;
Serial.print(b); //виведе до порту рядок «79»
```

```
Serial.println()
```

Функція `Serial.println` аналогічна функції `Serial.print`, і має такі ж варіанти виклику. Єдина відмінність полягає в тому, що після даних додатково виводяться два символи – символ повернення каретки (ASCII 13, або `\r`) і символ нової лінії (ASCII 10, або `\n`).

Приклад 1 (рис 4.2) та приклад 2 (рис. 4.3) виведе до порту одне і те саме. Підключіть плату Arduino до ПК та по черзі завантажте приклади та подивіться на результат роботи програм натиснувши кнопку  на панелі інструментів.

Робота з монітором послідовного порту в Processing ідентична як і в Arduino.

Приклади програм роботи функцій Serial у Processing дивіться у бібліотеці прикладів: File->Examples->Libraries->Serial.

```
int b = 79;

void setup() {
  Serial.begin(4800);
}

void loop() {
  Serial.print(b, DEC); //виведе до порту строку «79»
  Serial.print("\r\n"); //виведе символи "\r\n" - перевід строки
  Serial.print(b, HEX); //виведе до порту строку «4F»
  Serial.print("\r\n"); //виведе символи "\r\n" - перевід строки
}
```

Рисунок 4.2 – Приклад №1

```
int b = 79;

void setup() {
  Serial.begin(4800);
}

void loop() {
  Serial.println(b, DEC); //виведе до порту «79\r\n»
  Serial.println(b, HEX); //виведе до порту «4F\r\n»
}
```

Рисунок 4.3 – Приклад №2

Якщо у вас декілька пристроїв підключених до послідовного порту ПК, то можливо при компіляції вам буде висвічуватись помилка стосовно послідовного порту. У цьому випадку спробуйте змінити

значення у `String portName = Serial.list()[0]`; з [0] на [1], або [2], це залежить від того скільки пристроїв підключено по ПК.

4.2 Завдання до виконання самостійної роботи

4.2.1 Ознайомитися з теоретичними відомостями.

4.2.2 Допрацюйте у Processing програму з колом, яке рухається, з самостійної роботи №2, щоб координати центра кола виводилися до монітору послідовного порту Processing.

4.2.3 В Arduino розробити програму, яка зчитує значення, яке вводяться користувачем у моніторі послідовного порту та відповідно до нього встановлює положення серво підключеного до плати Arduino.

4.2.4 Оформити звіт до самостійної роботи.

4.3 Контрольні запитання

4.3.1 Назвіть основні команди для роботи з Serial Monitor?

4.3.2 Яка відмінність між `print` та `println`?

4.3.3 З якою швидкістю передаються данні за замовчанням в Arduino?

4.3.4 Які стандартні швидкості передачі даних ви знаєте?

4.4 Зміст звіту

4.4.1 Мета самостійної роботи.

4.4.2 Короткі теоретичні відомості.

4.4.3 Код програми.

4.4.4 Знімки екрану, які відображають виконані завдання.

4.4.5 Висновки до роботи.

5 САМОСТІЙНА РОБОТА 5

ВЗАЄМОДІЯ PROCESSING ТА ARDUINO

Мета роботи: навчитися створювати у Processing IDE простий графічний інтерфейс який за допомогою послідовного порту «спілкується» з Arduino.

5.1 Теоретичні відомості

Оскільки Processing та Arduino IDE тісно пов'язані між собою, а саме, перший є нащадком другого, то вони можуть взаємодіяти, що дозволяє створити простий графічний інтерфейс для зручного керування Arduino платформою або навпаки.

Дані можуть передаватися між ескізом Processing і платою Arduino за допомогою порту послідовної взаємодії. Для роботи з послідовним портом у Arduino і Processing використовується та ж сама бібліотека Serial, що і в попередній самостійній роботі.

В нижче наведених прикладах ми реалізуємо прості програми для демонстрації двостороннього зв'язку з Arduino до Processing і в зворотному напрямку.

Також існує можливість безпосередньо з Processing IDE спілкуватися та задавати команди платформі Arduino, для цього використовується протокол Firmata, який ми будемо розглядати у наступній роботі.

Підключить кнопку до плати Arduino, як зображено на рисунку 5.1. У схемі використовується резистор 10кОм.

Ознайомтесь з кодом який наведений нижче (рис 5.2), скопіюйте, відкомпілюйте та завантажте програму до плати Arduino.

Далі відкрийте середовище Processing, скопіюйте програму (рис 5.3), яка наведена нижче у прикладі та збережіть програму на диску комп'ютера (**File -> Save**).

Клікніть на панелі інструментів кнопку **Run (Запуск)**, це запустить компіляцію програми.

Якщо в вас декілька пристроїв підключених до послідовного порту ПК, то можливо при компіляції вам буде висвічуватись помилка

стосовно послідовного порту. У цьому випадку спробуйте змінити значення у `String portName = Serial.list()[0];` з `[0]` на `[1]`, або `[2]`, це залежить від того скільки пристроїв підключено по ПК.

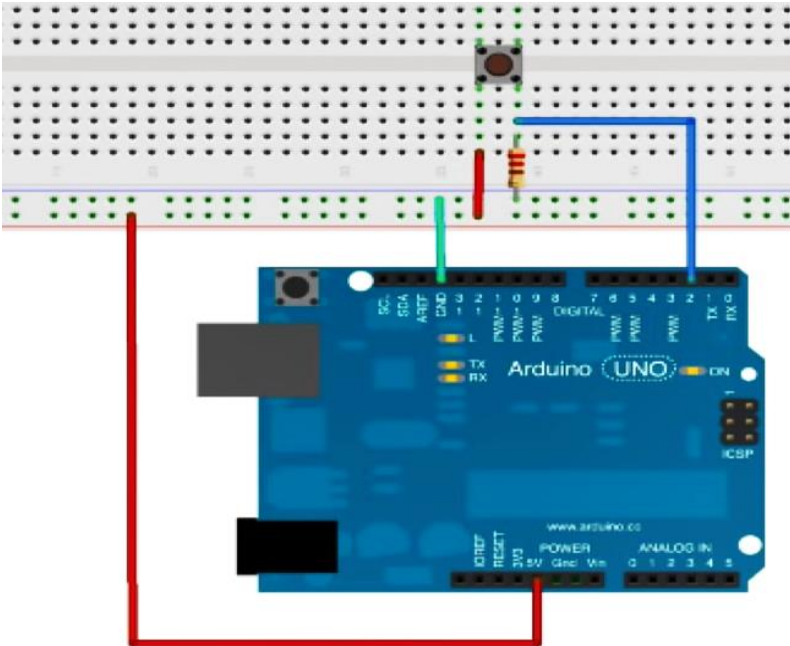


Рисунок 5.1 – Схема підключення кнопки до плати Arduino

У результаті ви побачте вікно відображення роботи Processing, в якому намальований квадрат розмірами 50 на 50 пікселів `rect(50, 50, 100, 100)`; верхній лівий кут якого має координати (50, 50), він середнього сірого кольору `fill(128)` (рис.5.4а). При натисканні кнопки квадрат буде змінювати свій колір з середнього сірого на помаранчевий `fill(255, 100, 0)`; (рис.5.4б).

```

int switchPin = 2; // кнопка підключена до 2 піну

void setup() {
  pinMode(switchPin, INPUT); // режим кнопки на вхід
  Serial.begin(9600); // розпочати послідовний зв'язок зі швидкістю 9600 біт за секунду
}

void loop() {
  if (digitalRead(switchPin) == HIGH) { // якщо кнопка зімкнута,
    Serial.write(1); // відправка 1 до Processing
  } else { // якщо кнопка розімкнута,
    Serial.write(0); // відправка 0 до Processing
  }
  delay(100); // затримка 100 мілісекунд
}

int switchPin = 2; // кнопка підключена до 2 піну

void setup() {
  pinMode(switchPin, INPUT); // режим кнопки на вхід
  Serial.begin(9600); // розпочати послідовний зв'язок зі швидкістю 9600 біт за секунду
}

void loop() {
  if (digitalRead(switchPin) == HIGH) { // якщо кнопка зімкнута,
    Serial.write(1); // відправка 1 до Processing
  } else { // якщо кнопка розімкнута,
    Serial.write(0); // відправка 0 до Processing
  }
  delay(100); // затримка 100 мілісекунд
}

```

Рисунок 5.2 – Код програми Arduino

Приклад взято з бібліотеки прикладів Processing.

```

import processing.serial.*;

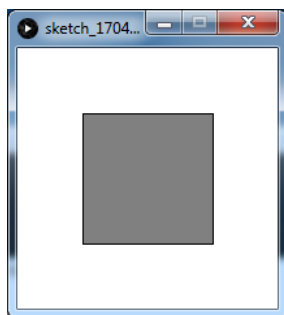
Serial myPort; // створює об'єкт класу Serial
int val; // змінна для збереження даних отриманих від Arduino

void setup()
{
    size(200, 200);
    String portName = Serial.list()[0];
    myPort = new Serial(this, portName, 9600);
}

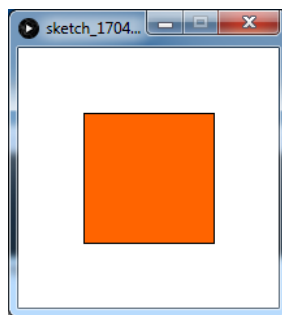
void draw()
{
    if ( myPort.available() > 0) { // якщо порт відкритий,
        val = myPort.read();      // читати дані з нього і зберегти у val
    }
    background(255); // встановити задній фон як білий
    if (val == 0) {   // якщо значення отримане з послідовного порту дорівнює 0,
        fill(128);    // встановити заповнення як середнє сіре
    }
    else {            // якщо значення відмінно від 0,
        fill(255, 100, 0); // встановити заповнення як помаранчеве
    }
    rect(50, 50, 100, 100); // прямокутник розміром 100 на 100
}

```

Рисунок 5.3 – Код програми Processing



а)



б)

Рисунок 5.4 – Результат роботи Processing

5.2 Завдання до виконання самостійної роботи

5.2.1 Ознайомитися з теоретичними відомостями.

5.2.2 Підключіть до Arduino RGB світлодіод. У Processing розробіть вікно з трьома квадратами різного кольору, щоб при натисканні на квадрат заданого кольору, параметри цього кольору передавалися до Arduino та світлодіод загорявся відповідним кольором.

5.2.3 Розробіть інтерфейс у Processing до завдання 4.2.3 самостійної роботи №4.

5.2.4 Оформити звіт до самостійної роботи.

5.3 Контрольні питання

5.3.1 Чому Arduino та Processing тісно пов'язані?

5.3.2 За допомогою чого передаються дані між Processing та Arduino?

5.3.3 Яка бібліотека використовується для роботи з послідовним портом у Arduino та Processing?

5.4 Зміст звіту

5.4.1 Мета самостійної роботи.

5.4.2 Короткі теоретичні відомості.

5.4.3 Код програми.

5.4.4 Знімки екрану, які відображають виконані завдання.

5.4.5 Висновки до роботи.

6 САМОСТІЙНА РОБОТА 6

РОЗРОБКА ГРАФІЧНОГО ІНТЕРФЕЙСУ З ВИКОРИСТАННЯМ БІБЛІОТЕКИ FIRMATA

Мета роботи: ознайомитись з бібліотекою Firmata для Arduino навчитися використовувати її для роботи з Processing IDE.

6.1 Теоретичні відомості

В бібліотеці Firmata реалізований однойменний протокол, призначений для зв'язку мікроконтролера з прикладним програмним забезпеченням на комп'ютері. Використання бібліотеки Firmata звільняє від необхідності написання власних протоколів і об'єктів для взаємодії пристрою і ПК.

Більш детально з функціями можна ознайомитись за посиланням <http://arduino.ua/ru/prog/Firmata>.

6.2 Завдання до виконання самостійної роботи

6.2.1 Ознайомитися з теоретичними відомостями.

6.2.2 Використовуючи бібліотеку Firmata для Arduino розробити програму:

1) Інтерфейс у вигляді повзунка (як для регулювання гучності на ПК), та з використанням потенціометра, підключеного до Arduino плати, зробити так, щоб з поворотом потенціометра повзунок переходив з одного краю до іншого.

2) Навпаки щоб перетягуючи повзунок з одного краю до іншого мінявся колір RGB світлодіоду, підключеного до Arduino плати.

6.2.3 Оформити звіт до самостійної роботи.

6.3 Контрольні питання

6.3.1 Розкажіть в чому полягає суть роботи бібліотеки Firmata?

6.3.2 Назвіть основні методи бібліотеки Firmata та їх призначення.

6.3.3 Назвіть методи бібліотеки Firmata, які використовуються для відправки повідомлень?

6.4 Зміст звіту

6.4.1 Мета самостійної роботи.

6.4.2 Короткі теоретичні відомості.

6.4.3 Код програми.

6.4.4 Знімки екрану, які відображають виконані завдання.

6.4.5 Висновки до роботи.

ДЖЕРЕЛА ІНФОРМАЦІЇ

1. Processing [Electronic source]. – Access mode: <https://processing.org/download/?processing>

2. Wiring. [Електронний ресурс]. – Режим доступу: <http://wiring.org.co/>

3. Arduino [Електронний ресурс]. – Режим доступу: <https://www.arduino.cc/>