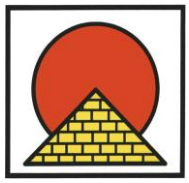
Регионални центар за талента
Михајло Пупин
Панчево



Processing приручник

Приручник за развој апликација
помоћу *Processing* библиотеке

Др Јован Поповић

2024

Увод

Овај приручник је направљен за потребе предавања у Центру за таленте РЦТ Михајло Пупин. Он представља материјале по којима ученици могу да се упознају са начинима имплементације *Processing* апликација. У овом приручнику су објашњени основни концепти као што су:

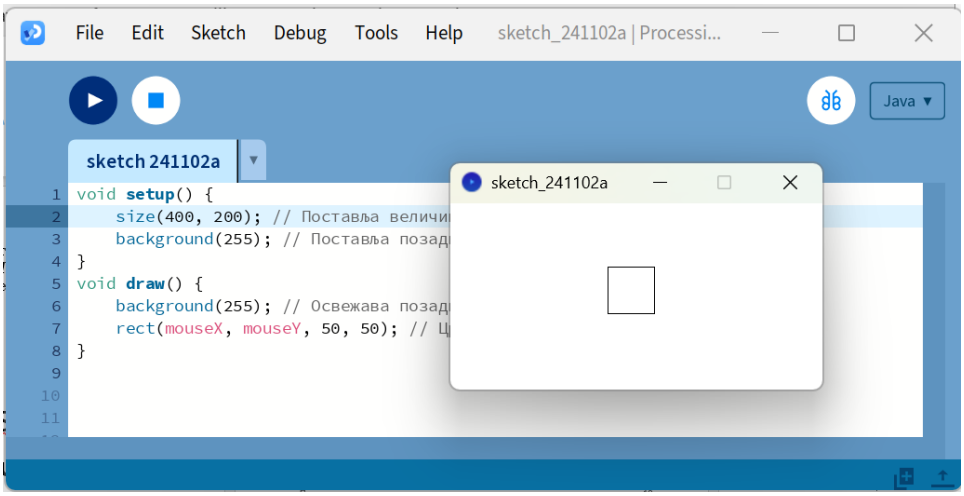
- Цртање фигура и слика: Научићете како да користите основне алате за цртање правоугаоника, кругова, линија и других облика. Такође ћете научити како да увозите и приказујете слике у вашој *Processing* апликацији.
- Анимације: Овај део обухвата креирање динамичких и интерактивних анимација. Научићете како да користите функције за анимацију, променљиве које контролишу кретање и како да направите глатке прелазе између различитих стања.
- Рад са мишем и тастатуром: У овом делу научићете како да обрађујете унос са миша и тастатуре како би ваше апликације постале интерактивне. Научићете како да детектујете кликове мишем, кретање миша, притиске тастера и како да реагујете на ове догађаје.
- Рад са камером: Упознаћете се са коришћењем камере у *Processing* апликацијама. Научићете како да приступите камери, снимате слике и радите са видео токовима у реалном времену.
- Рад са серијским портом: Овај део ће вам показати како да комуницирате са спољним уређајима преко серијског порта. Научићете како да шаљете и примате податке, као и како да обрадите те податке у вашој *Processing* апликацији.

Помоћу овог приручника ћете бити спремни да успешно користите *Processing* окружење.

Садржај

Шта је *Processing* окружење?

Processing је интегрисано развојно окружење (IDE) и библиотека који је развијен како да би олакшао креирање графичких апликација, анимација и интерактивних уметничких дела. *Processing* окружење је једноставно за коришћење и омогућава брз развој визуелних апликација. Састоји се од уређивача кода (скеча), конзоле за извршавање и приказивача резултата.



Програмски код можете написати на неком програмском језику као што је Java, Python или JavaScript коришћењем функција које су прилагођене за лакше креирање графичких објеката и анимација.

Processing апликација се може бесплатно преузети са сајта https://processing.org/download.

У *Processing*-у, корисници могу користити једноставан Јава код за креирање цртежа, графичких интерфејса и анимација. Основне предности ових апликација су:

1. **Цртање**: *Processing* омогућава лако цртање облика као што су линије, правоугаоници, елипсе и други. На пример, функција **rect(x, y, ширина, висина)** црта правоугаоник на задатим координатама.
2. **Графички интерфејс**: Могуће је креирати интерактивне графичке интерфејсе користећи миш и тастатуру. На пример, функција **mousePressed()** може детектовати када је корисник кликнуо мишем и извршити одређени код.
3. **Анимације**: *Processing* подржава креирање анимација коришћењем функције **draw()**, која се стално извршава како би се нацртао нови фрејм анимације.

Корисници могу ажурирати позиције објеката и поново их цртати како би створили ефекат кретања.

Ево једноставног примера кода који црта правоугаоник и анимира га:

```
void setup() {
  size(800, 600); // Поставља величину екрана на 800x600
}
void draw() {
  rect(mouseX, mouseY, 50, 50); // Црта правоугаоник на позицији миша
}
```

У овом примеру, функција **setup()** се користи за иницијализацију, док се функција **draw()** стално извршава и црта правоугаоник на позицији миша, стварајући ефекат анимације.

Костур Processing програма

Када пишемо програм у Processing-у, користимо две основне методе: **setup()** и **draw()**. Ове методе омогућавају нам да иницијализујемо програм и цртамо различите елементе на екрану. У следећем листингу је приказан основни код апликације:

```
void setup() {
  // Овде иде код који се извршава само при покретању програма
}
void draw() {
  // Овде иде код за цртање којим се црта нови фрејм на екрану
}
```

Функција **setup()** се извршава само једном, приликом покретања програма. Ова функција се користи за подешавање иницијалних параметара програма, као што су величина прозора, позадина или учитавање података који ће бити приказани.

Функција **draw()** метода се извршава континуирано, при чему се сваки пут црта нови фрејм. Све што је нацртано у овој методи ће бити приказано на екрану. Ова функција се користи за динамичко цртање и анимације.

Овде смо описали основни костур програма у *Processing*-у и објаснили функције **setup()** и **draw()**. Овај костур представља полазну тачку за креирање различитих интерактивних програма и анимација у *Processing*-у.

Функција setup()

Функција **setup()** се извршава само једном на почетку програма. Користи се за иницијализацију подешавања, као што су постављање величине екрана, подешавање почетних вредности променљивих и других параметара – на пример:

```
void setup() {
  size(800, 600); // Поставља величину екрана на 800x600
  background(255); // Поставља позадину на белу боју
}
```

У овом примеру, функција **setup()** поставља величину екрана на 800x600 пиксела и подешава позадину на белу боју. У *Processing*-у, постоји неколико функција које се могу користити унутар функције **setup()** за подешавање различитих параметара као што су величина екрана, боја позадине, боја оловке, ширина оловке и слично. Ево неких од најчешће коришћених функција:

1. **size(width, height) -** Ова функција поставља величину прозора за приказивање. На пример **size(800, 600)** поставља величину екрана на 800 пиксела ширине и 600 пиксела висине.
2. **background(value) -** Ова функција поставља боју позадине. Може се користити један параметар за нијансу сиве или три параметра за RGB вредности. На пример **background(255)** поставља позадину на белу боју, а **background(0, 0, 255)** поставља позадину на плаву боју.
3. **stroke(color)** Ова функција поставља боју оловке (линија и ивица облика). На пример **stroke(0)** поставља боју оловке на црну, док **stroke(255, 0, 0)** поставља боју оловке на црвену.
4. **strokeWeight(weight) -** Ова функција поставља ширину оловке. На пример **strokeWeight(2)** поставља ширину оловке на 2 пиксела.
5. **noStroke() -** Ова функција уклања оловку, тако да облици немају ивице. На пример **noStroke()** уклања ивице са свих облика.
6. **fill(color) -** Ова функција поставља боју попуне за облике. На пример **fill(150)** поставља боју попуне на сиву, док **fill(0, 255, 0)** поставља боју попуне на зелену.
7. **noFill() -** Ова функција уклања попуну из облика. На пример **noFill()** уклања попуну из свих облика.

8. **scale()**: Ова функција се користи за скалирање координатног система. Може се користити за увећање или умањење објеката на екрану. На пример, позивањем **scale(2)** сви објекти ће бити дупло већи, док позивањем **scale(1.5, 0.5)** објекти ће бити скалирани 1.5 пута по x оси и 0.5 пута по y оси.
9. **translate()**: Ова функција се користи за померање координатног система. Омогућава померање почетне тачке координатног система на нову позицију. На пример, позивањем **translate(200, 150)** координатни систем ће бити померен за 200 пиксела десно и 150 пиксела доле, а све наредне координате ће бити релативне у односу на нову почетну тачку.

Коришћење ових функција у **setup()** функцији је приказан у следећем листингу:

```
void setup() {
  size(800, 600); // Поставља величину екрана на 800x600
  background(255); // Поставља позадину на белу боју
  stroke(0); // Поставља боју оловке на црну
  strokeWeight(2); // Поставља ширину оловке на 2 пиксела
  fill(0, 255, 0); // Поставља боју унутрашњости фигура на зелену
  translate(width / 2, height / 2); // Поставља центар на половину екрана
  scale(1, -1); // Окреће вертикалну осу на горе
}
```

У овом примеру, функција **setup()** поставља величину екрана, боју позадине, боју и ширину оловке, и боју попуне за облике. Функције **translate()** и **scale()** омогућавају постављање координатног система тако да је центар на средини екрана, а вертикална оса окренута на горе. Ове функције омогућавају флексибилно подешавање изгледа и понашања графичких елемената у *Processing* програму.

Функција draw()

Функција **draw()** се стално извршава све док програм ради. Користи се за цртање и ажурирање графике на екрану. Све што се стави унутар ове функције ће се извршавати континуирано, омогућавајући креирање анимација и интерактивних елемената. На пример:

```
void draw() {
  rect(mouseX, mouseY, 50, 50); // Црта правоугаоник на позицији миша
}
```

У овом примеру, функција **draw()** на сваком фрејму црта правоугаоник на позицији миша, стварајући ефекат анимације.

Цртање облика

Processing библиотека обезбеђује велики број функција којима се могу цртати различити геометријски облици:

- **line(x1, y1, x2, y2)** - Црта линију од тачке (x1, y1) до тачке (x2, y2). На пример **line(50, 50, 150, 150)** црта линију од координате (50, 50) до координате (150, 150).
- **rect(x, y, ширина, висина)** - Црта правоугаоник са горњим левим углом на координатама (x, y), ширине ширина и висине висина. На пример **rect(100, 100, 200, 150)** црта правоугаоник са горњим левим углом на координатама (100, 100), ширине 200 и висине 150 пиксела.
- **ellipse(x, y, ширина, висина)** - Црта елипсу са центром на координатама (x, y), ширине ширина и висине висина. На пример: **ellipse(200, 200, 100, 50)** црта елипсу са центром на координатама (200, 200), ширине 100 и висине 50 пиксела.
- **triangle(x1, y1, x2, y2, x3, y3)** - Црта троугао са теменима на координатама (x1, y1), (x2, y2) и (x3, y3). На пример **triangle(300, 100, 350, 200, 250, 200)** црта троугао са теменима на координатама (300, 100), (350, 200) и (250, 200).
- **quad(x1, y1, x2, y2, x3, y3, x4, y4)** - Црта четвороугао са теменима на координатама (x1, y1), (x2, y2), (x3, y3) и (x4, y4). На пример **quad(100, 100, 200, 100, 250, 200, 150, 200)** црта четвороугао са теменима на координатама (100, 100), (200, 100), (250, 200) и (150, 200).
- **arc(x, y, ширина, висина, почетни_угао, крајњи_угао)** - Црта лук елипсе са центром на координатама (x, y), ширине ширина и висине висина, од почетни_угао до крајњи_угао. На пример **arc(300, 300, 100, 100, 0, PI)** црта полукруг са центром на координатама (300, 300), ширине и висине 100 пиксела, од 0 до π радијана.
- **point(x, y)** - Црта тачку на координатама (x, y). На пример **point(400, 400)** црта тачку на координатама (400, 400).

Ове функције омогућавају креирање различитих облика у *Processing*-у, што је основа за развој графичких апликација и анимација.

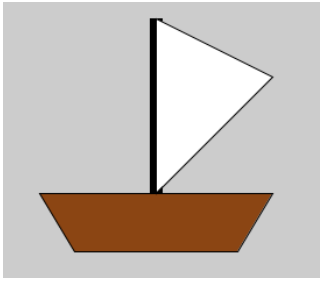
Processing омогућава да се направе и сложенији многоуглови тако што се редом задају тачке многоугаоне линије коришћење, функција **beginShape()**, **vertex()** и **endShape()**.

- Почетак дефиниције облика - **beginShape()**: Функција **beginShape()** започиње цртање сложенијег облика. Унутар ове функције ћемо додавати тачке које ће чинити стране нашег многоугла. На пример, почињемо облик са **beginShape()**.

- Додавање тачака - **vertex()**: Функција **vertex(x, y)** дефинише појединачне врхове (тачке) облика. Сваки позив функције **vertex()** додаје нову тачку на форму. Тачке су редом повезане линијама, стварајући стране многоугла. На пример, додајемо тачке са **vertex(100, 100)**, **vertex(150, 50)**, и **vertex(200, 100)**.
- Затварање облика - **endShape()**: Функција **endShape()** завршава дефиницију облика. Ако желимо да наш многоугао буде затворен (почетна и завршна тачка да буду повезане), можемо користити параметар **CLOSE**. На пример, затварамо облик са **endShape(CLOSE)**.

Пример – брод

У следећем примеру можемо да видимо код који црта брод у *Processing*-у користећи неколико различитих облика и боја да би створио слику брода. Овај код користи различите облике и боје да би нацртао брод. Труп брода се црта као правоугаоник, платно као троугао, јарболи као правоугаоници, и лукови представљају ветар у једрима



```
void boat(float x, float y) {
  // Труп брода (правоугаони трапез)
  fill(139, 69, 19); // Смеђа боја
  beginShape();
  vertex(x - 100, y);
  vertex(x + 100, y);
  vertex(x + 70, y + 50);
  vertex(x - 70, y + 50);
  endShape(CLOSE); // Трапезни труп брода

  // Јарбол (правоугаоник)
  fill(0); // Црна боја
  rect(x - 5, y - 150, 10, 150); // Јарбол

  // Платно (једро)
  fill(255); // Бела боја
  triangle(x, y - 150, x, y, x + 100, y - 100); // Једро наслоњено на јарбол
}
```

Пример – смајли

У следећем примеру је приказана функције која црта смајлија на координати (x,y):

```
void smiley(float x, float y) {
  // Глава (кружница)
  fill(255, 204, 0); // Жута боја
  ellipse(x, y, 200, 200); // Кружница за главу

  // Лево око (два круга)
  fill(255); // Бела боја за беоњачу
  ellipse(x - 50, y - 50, 40, 40); // Спољашњи круг за лево око
  fill(0); // Црна боја за зеницу ока
  ellipse(x - 50, y - 50, 20, 20); // Унутрашњи круг за лево око

  // Десно око (два круга)
  fill(255); // Бела боја за беоњачу
  ellipse(x + 50, y - 50, 40, 40); // Спољашњи круг за беоњачу
  fill(0); // Црна боја за зеницу
  ellipse(x + 50, y - 50, 20, 20); // Унутрашњи круг за зеницу

  // Нос (правоугаоник)
  fill(255, 0, 0); // Црвена боја
  rect(x - 10, y - 20, 20, 40); // Правоугаоник за нос

  // Уста (лук)
  noFill(); // Без попуне
  stroke(0); // Црна боја за ивицу
  strokeWeight(5); // Ширина ивице
  arc(x, y + 20, 120, 60, 0, PI); // Лук за уста
}
```

Цртање слика

У Processing-у, слике се могу додавати и приказивати унутар **draw()** методе користећи функцију **image()**. Пре него што се слика може приказати, потребно је да се учита у меморију помоћу функције loadImage() у setup() методи. Ево корака како се то ради:

1. **Учитавање слике у setup() методи**:Користи се функција **loadImage()** за учитавање слике из датотеке. Пример: PImage img = loadImage("ime_slike.jpg");
2. **Приказивање слике у draw() методи**: Користи се функција **image()** за приказивање учитане слике на одређеним координатама. Пример: image(img, x, y);

Пример кода је приказан у следећем листингу.

```
PImage img; // Декларација променљиве за слику

void setup() {
  size(800, 600); // Поставља величину екрана на 800x600
  img = loadImage("ime_slike.jpg"); // Учитава слику из датотеке
}

void draw() {
  background(255); // Освежава позадину
  image(img, 100, 100); // Приказује слику на координатама (100, 100)
}
```

У овом примеру, слика се учитава у **setup()** методи помоћу **loadImage()** и чува у променљивој **img**. У **draw()** методи, слика се приказује на координатама (100, 100) помоћу функције **image()**.

Додатне опције за image()

Функција **image()** има и додатне параметре за скалирање слике. Позивом функције са додатним параметрима **image(img, x, y, ширина, висина)** приказује се слика на координатама (x, y) са задатом ширином и висином. На пример **image(img, 100, 100, 200, 150)** приказује слику на координатама (100, 100) са ширином 200 и висином 150 пиксела.

Исписивање текста

Исписивање текста у *Processing* апликацији је једноставно и омогућава вам да прикажете различите информације на екрану. Основне функције за испис текста су:

- **text()** функција се користи за исписивање текста. Прихвата три параметра: текст који желите да прикажете, и x и y координате на којима желите да прикажете текст.
- **textSize()** функција поставља величину текста. Прихвата један параметар који одређује величину фонта у пикселима.
- **fill()** функција поставља боју текста. Прихвата један до три параметра: црвену, зелену и плаву компоненту боје (RGB).
- **textAlign()** функција поставља поравнање текста. Прихвата два параметра: поравнање хоризонтално (**LEFT**, **CENTER**, **RIGHT**) и вертикално (**TOP**, **CENTER**, **BOTTOM**).

Пример - испис текста

Ево примера кода који приказује како се користе ове функције за испис текста:

```
void setup() {
  size(800, 600); // Величина канваса
  background(255); // Бела позадина

  // Постављање величине текста
  textSize(32);

  // Постављање боје текста
  fill(0, 102, 153);

  // Постављање поравнања текста
  textAlign(CENTER, CENTER);

  // Исписивање текста
  text("Здраво, свете!", width / 2, height / 2);
}
```

У овом примеру, текст "Здраво, свете!" ће бити приказан у центру са величином фонта 32, бојом текста плавом (0, 102, 153) и централним поравнањем.

Анимације

Анимације у *Processing*-у омогућавају креирање динамичких и интерактивних визуелних приказа. Да бисмо направили анимацију, потребно је дефинисати променљиве које представљају координате објекта који се креће, нацртати тај објекат на тим координатама и у **draw()** методи мењати вредности координата за сваки фрејм.

Основни кораци за креирање анимације:

1. Дефинисање променљивих: Прво, дефинишемо променљиве које ће представљати координате објекта. Ове променљиве ће се мењати у сваком фрејму да би се симулирало кретање.

```
float x; // Координата објекта
```

2. Иницијализација у **setup()** методи: У setup() методи постављамо иницијалне вредности координата и подешавамо величину прозора.

```
void setup() {
  size(800, 600); // Величина прозора
  x = 0; // Почетна координата
}
```

3. У **draw()** методи цртамо објекат на основу координата и мењамо вредности координата за сваки фрејм.

```
  void draw() {
    background(255); // Поставља белу позадину
    ellipse(x, height/2, 50, 50); // Црта круг на координати и половини висине
    x += 1; // Повећава координату за 1 у сваком фрејму
  }
```

ВАЖНО: Коришћење **background()** функције у **draw()** методи је веома важно. Сваки позив **draw()** методе представља један фрејм анимације. Без **background()** функције, сваки нови фрејм би се цртао преко претходног, што би довело до ефекта да се сви фрејмови преклапају. **background()** функција поставља позадину пре исцртавања фрејма, чиме "брише" све што је било нацртано у претходном фрејму. Ово омогућава да сваки нови фрејм почне са чистом позадином, тако да можемо јасно видети нову позицију објеката у анимацији.

Пример – коси хитац

Ево примера кода за анимацију косог хица у Processing-у. Круг ће се налазити на позицији (50, 50) са брзинама **vx** и **vy**, а гравитација и временски интервал у коме ћемо мењати ов вредности и померати објекат ће бити дефинисани као **g** = 8.91 и **dt** = 1. Нове координате и брзине ће се рачунати у свакој итерацији:

```
// Дефинисање променљивих
float x = 50;
float y = 50;
float vx = 5; // Почетна хоризонтална брзина
float vy = -10; // Почетна вертикална брзина
float g = 8.91; // Гравитација
float dt = 1; // Временски интервал

void setup() {
  size(800, 600); // Величина канваса
}

void draw() {
  background(255); // Бела позадина

  // Цртање круга
  ellipse(x, y, 20, 20);

  // Рачунање нових координата и брзина
  x = x + vx*dt;
  y = y + vy*dt;
  vy = vy + g*dt;

  // Превенција изласка круга изван канваса (опционо)
  if (y > height) {
    y = height;
    vy = -vy * 0.7; // Смањује брзину при одскоку
  }
}
```

Круг који се креће по трајекторији косог хица. Гравитација ће утицати на вертикалну брзину (vy), док ће хоризонтална брзина (vx) остати константна. Круг ће одскочити када додирне дно, симулирајући одскок са смањеном брзином.

Интеракција са мишем и тастатуром

Коришћење миша

У *Processing*-у, можемо да идентификујемо неку интеракцију мишем на два начина:

- Помоћу променљивих и у којима су координате тренутне позиције миша.
- Помоћу функција које позива када се нешто деси са мишем (нпр. када се помери притисне дугме на мишу и слично).

Координате миша можемо добити помоћу променљивих **mouseX** и **mouseY**. Ове променљиве увек садрже тренутну позицију миша. Ево једног примера где се круг (лопта) помера на координате миша у сваком фрејму:

```
void setup() {
  size(800, 600); // Величина канваса
}
void draw() {
  background(255); // Бела позадина
  ellipse(mouseX, mouseY, 50, 50); // Црта круг на тренутним координатама
}
```

У *Processing*-у, можемо реаговати на различите догађаје мишем помоћу функција које се позивају као „евент хендлери“. Ове функције омогућавају нам да детектујемо када је миш померен, када је кликнуто дугме или када је дугме отпуштено. Ево неких од основних функција којима се реагује како би се реаговало на догађаје који се десе мишем:

- **mousePressed()** – ова функција се позива када се било које дугме миша притисне. Може се користити за извршавање кода сваки пут када је миш притиснут.

```
void mousePressed() {
  // Код који се извршава када је дугме миша притиснуто
}
```

- **mouseReleased()** – ова функција се позива када се било које дугме миша отпусти. Може се користити за извршавање кода сваки пут када је миш отпуштен.

```
void mouseReleased() {
  // Код који се извршава када је дугме миша отпуштено
}
```

- **mouseMoved()** – ова функција се позива сваки пут када се миш помери. Може се користити за детектовање кретања миша.

```
void mouseMoved() {
  // Код који се извршава када се миш помери
}
```

- **mouseDragged()** – ова функција се позива када се миш помера док је неко дугме притиснуто.

```
void mouseDragged() {
    // Код који се извршава када се миш помера док је дугме притиснуто
}
```

- **mouseClicked()** – ова функција се позива када се дугме миша притисне и отпусти. Може се користити за извршавање кода сваки пут када је миш кликнут.

```
void mouseClicked() {
  // Код који се извршава када је миш кликнут
}
```

- **mouseWheel()** – ова функција се позива када се точкић миша ротира. Може се користити за детектовање померања точкића миша горе или доле.

```
void mouseWheel(MouseEvent event) {
    float e = event.getCount(); // Враћа вредност померања точкића (позитивно за доле, негативно за горе)
    // Код који се извршава када се точкић миша ротира
}
```

Пример – померање лопте мишем

Ево примера који користи **mousePressed()**, **mouseClicked()** и **mouseWheel()** функције за померање лопте и промену њене величине на основу догађаја са мишем:

```
float ballX;
float ballY;
float ballSize = 50;

void setup() {
  size(800, 600); // Величина канваса
  ballX = width / 2; // Почетна координата x
  ballY = height / 2; // Почетна координата y
}

void draw() {
  background(255); // Бела позадина
  ellipse(ballX, ballY, ballSize, ballSize); // Црта круг на тренутним координатама
}

void mousePressed() {
  ballX = mouseX; // Поставља x координату круга на x координату миша
  ballY = mouseY; // Поставља y координату круга на y координату миша
}

void mouseClicked() {
  fill(random(255), random(255), random(255));
  // Промена боје круга на случајну боју при сваком клику
}

void mouseWheel(MouseEvent event) {
  float e = event.getCount(); // Враћа вредност померања точкића
  ballSize += e * 5; // Промена величине круга на основу померања точкића
  if (ballSize < 10) ballSize = 10; // Минимална величина круга
}
```

У овом примеру, **mousePressed()** поставља координате круга на позицију миша која је дефинисана променљивама **mouseX** и **mouseY**, **mouseClicked()** мења боју круга на случајну боју при сваком клику, а **mouseWheel()** мења величину круга на основу ротације точкића миша. Ово омогућава интерактивну анимацију која реагује на различите догађаје са мишем.

Тастатура

Рад са тастатуром у *Processing*-у омогућава нам да створимо интерактивне програме који реагују на унос са тастатуре. Можемо детектовати притиске тастера и извршавати код на основу унетих тастера. Ево основних функција којима се реагује на притиснуте тастере:

- **keyPressed()** – ова функција се позива када се било који тастер на тастатури притисне. Може се користити за извршавање кода сваки пут када је тастер притиснут, укључујући и стрелице или функционалне тастере (**Shift**, **Ctrl**, итд.) .

```
void keyPressed() {
  // Код који се извршава када је тастер притиснут
}
```

- **keyTyped()** – ова функција се позива сваки пут када се притисне тастер који уноси карактер. Може се користити за извршавање кода сваки пут када је унет карактер. За разлику од ова функција се позива само када је притиснут тастер који уноси карактер (нпр. слова, бројеви). Она не детектује специјалне тастере као што су стрелице или функционални тастери (**Shift**, **Ctrl**, итд.).

```
void keyTyped() {
  // Код који се извршава када је унет карактер
}
```

- **keyReleased()** – ова функција се позива када се било који тастер на тастатури отпусти. Може се користити за извршавање кода сваки пут када је тастер отпуштен.

```
  void keyReleased() {
    // Код који се извршава када је тастер отпуштен
  }
```

У *Processing*-у можемо користити промењиве **key** и **keyCode** за добијање информација о притиснутом тастеру. Промењива **key** представља карактер притиснутог тастера, док **keyCode** представља код специјалних тастера као што су стрелице, **Shift**, **Control** итд.

У следећој табели су приказане вредности које имају ове променљиве за неке карактере:

Карактер	**key вредност**	**keyCode вредност**
A-Z	'A'-'Z'	65-90
a-z	'a'-'z'	97-122
0-9	'0'-'9'	48-57
Space (размак)	' '	32
Enter (Унос)	'\n'	ENTER
Backspace (Брисање)	'\b'	BACKSPACE
Tab	'\t'	TAB
Escape	27	ESC
Up, Down, Left, Right Arrow	N/A	UP, DOWN, LEFT, RIGHT
Page Up, Page Down	N/A	PAGE_UP, PAGE_DOWN
Shift, Ctrl, Alt	N/A	SHIFT, CONTROL, ALT
Home, End, Insert, Delete	N/A	HOME, END, INSERT, DELETE
F1-F12	N/A	112-123

Ова табела вам омогућава да брзо и лако идентификујете **key** и **keyCode** вредности за обичне карактере, специјалне тастере и функционалне тастере у **Processing**-у. На пример, ако желите да детектујете притисак тастера Page Up, користите keyCode == **PAGE_UP**, или ако желите да детектујете притисак тастера **Home**, користите keyCode == HOME.

Пример – померање лопте стрелицама на тастатури

Ево једног примера где се круг помера по екрану у зависности од притиснутих стрелица на тастатури:

```
float x = 400;
float y = 300;

void setup() {
  size(800, 600); // Величина канваса
}

void draw() {
  background(255); // Бела позадина
  ellipse(x, y, 50, 50); // Цртање круга на тренутним координатама
}

void keyPressed() {
  if (keyCode == UP) {
    y -= 5; // Померање круга нагоре
  } else if (keyCode == DOWN) {
    y += 5; // Померање круга наниже
  } else if (keyCode == LEFT) {
    x -= 5; // Померање круга лево
  } else if (keyCode == RIGHT) {
    x += 5; // Померање круга десно
  }
}
```

У овом примеру, круг се помера горе, доле, лево или десно у зависности од притиснуте стрелице. Функција **keyPressed()** проверава који је тастер притиснут и мења координате круга у складу са тим. Ово омогућава једноставну интеракцију са програмом користећи тастатуру.

Коришћење ових функција и променљивих омогућава нам да креирамо различите врсте интерактивних програма у *Processing*-у који реагују на унос са тастатуре, што значајно повећава функционалност и корисничко искуство програма.

Коришћење камере

Код омогућава коришћење камере што укључује приказивање онога што снима камера и снимање слика са камере.

Први корак је да увезете видео библиотеку **processing.video** у код и дефинишете објекат који представља камеру:

```
import processing.video.*;
Capture cam;
```

Иницијализујте и покрените камеру у **setup()** методи. Поставите жељену резолуцију и изаберите камеру коју желите да користите.

```
void setup() {
  size(800, 600); // Величина прозора
  cam = new Capture(this, 640, 480); // Иницијализација камере са резолуцијом 640x480
  cam.start(); // Покретање камере
}
```

У **draw()** методи може да се провери да ли је нови фрејм доступан, и ако јесте, може се учитати слика коју тренутно снима камера која се може и приказати.

```
void draw() {
  if (cam.available() == true) {
    cam.read(); // Учитава нови фрејм из камере
  }
  image(cam, 0, 0); // Приказује слику са камере на канвасу
}
```

Комплетан пример кода који укључује све горе наведене кораке је представљен у следећем листингу:

```
import processing.video.*;
Capture cam;

void setup() {
  size(800, 600); // Величина прозора
  cam = new Capture(this, 640, 480); // Иницијализација камере са резолуцијом 640x480
  cam.start(); // Покретање камере
}

void draw() {
  if (cam.available() == true) {
    cam.read(); // Учитава нови фрејм из камере
  }
  image(cam, 0, 0); // Приказује слику са камере на канвасу
}

void keyPressed() {
  if (key == 's' || key == 'S') {
    saveFrame("snapshot-####.png"); // Снимање слике као PNG датотека
  }
}
```

Овај код користи камеру за приказивање видеа и омогућава снимање слике притиском на тастер 's'. На овај начин можете лако укључити камеру у свој *Processing* пројекат и снимати слике.

Прихватање података серијским портом

Processing библиотека има скуп класа и функција којима се могу читати и слати подаци путем серијског порта. Ова функционалност се користи када неки микроконтролер као на пример *Arduino* или *Microbit* шаље податке путем USB кабла. Микроконтролер који је повезан на порт може да мери неке вредности и шаље их рачунару преко конекције. На овај начин можете да мериве температуру, притисак очитавате вредности ултразвучног сензора и те податке шаљете рачунару како би биле приказане на неком графику.

Потребно је додати следећи програмски код у код како би се користиле функционалности за комуникацију са серијским портом.

```
import processing.serial.*;
Serial arduinoPort;
```

Објекат ***arduinoPort*** представља серијски порт којим се комуницира са микро-контролером. У методи ***setup()*** је потребно иницијализовати овај објекат, кодом који је приказан у следећем примеру:

```
arduinoPort = new Serial(this, Serial.list()[0], 9600);
arduinoPort.bufferUntil('\n');
```

У овом примеру серијски порт чита податке са нултог прикључка на који наиђе али у зависности од броја прикључака на рачунару можда ћете морати да промените број. Број 9600 је тзв. ***baud rate*** који представља брзину којом *Arduino* шаље податке. У случају да се користи неки други микроконтролер као на пример *Microbit*, потребно је користити његов као на пример 115200. Објекат ***arduinoPort*** који представља порт садржи методе којима се врши комуникације путем серијског порта:

- ***available()*** којом се проверава колико има података који чекају на серијском порту како би их неко прочитао.
- ***readStringUntil()*** којом се чита текст са порта све до знака који је прослеђен као параметар. У случају да микро-контролер шаље податке који су одвојени знаком за нови ред или запетом,
- ***write()*** којом се шаљу подаци ка *Arduino* уређају преко серијског порта.

Ове функције се могу користити у функцији ***draw()*** како би се прочитала вредност коју је послао микро-контролер:

```
 if ( arduinoPort.available() > 0)  {
   int val = Float.parseFloat(arduinoPort.readStringUntil('\n'));
   ...
}
```

Овакав код се може ставити у функцију ***draw()*** и иза њега се могу додати наредбе којима се вредност црта на неком графику. Уместо експлицитне провере да ли постоје подаци на порту, може се дефинисати функција која се активира сваки пот када се преко потра пошаље нови податак.

```
void serialEvent(Serial p) {
 val = p.readString();
}
```

Ова функција се позива сваки пут када се појави неки податак на порту и у њој се могу прочитати подаци који долазе преко порта. У неким апликацијама је лакше ставити код који чека и реагује на појаву нових података уместо да се стално у функцији ***draw()*** проверава да ли су подаци на располагању.

Пример – приказивање резултата мерења

Претпоставимо да уређај мери температуру сваке секунде и шаље мерене вредности одвојене новим редом преко серијског порта. Потребно је написати апликацију помоћу *Processing* библиотеке која ће читати те вредности и приказивати их на графику.

Као први корак је потребно импортовати све објекте из пакета и креирати објекат који представља серијски порт којим се комуницира са уређајем. У ***setup()*** функцији се дефинишу димензије апликације, помера координатни почетак на доњи леви угао (функцијама ***scale()*** и ***translate()***), цртају осе координатног система (са троугловима на крајевима који представљају стрелице) и иницијализује порт којим ће се од *Arduino* уређаја добијати информације.

```
void setup() {
 size(800, 600);

 // Померање координатном система у доњи леви угао
 scale(1,-1);
 translate(10,10-height);
```

```
 // Хоризонтална x-оса
 line(0,0, width-25, 0);
 triangle(width-25,5, width-25,-5, width-15,0);

 // Вертикална y-оса
 line(0,0, 0, height-25);
 triangle(5,height-25, -5,height-25, 0,height-15);

 arduinoPort = new Serial(this, Serial.list()[0], 9600);
 arduinoPort.bufferUntil('\n');
}
```

Сада је потребно додати код у функцију ***draw()*** који ће приликом сваког позива проверавати има ли нове вредности на серијском улазу. Ако је има, онда се та вредност чита и црта на графику.

```
int x = 1;

void draw()
{
 if ( arduinoPort.available() > 0)  {
  float val = Float.parseFloat(arduinoPort.readStringUntil('\n'));

  scale(1,-1);
  translate(10,10-height);

  stroke(0);
  line(x,1, x,val);
  stroke(255);
  line(x,val, x,height-val-35);

  x= x+1;
  if(x>=width-25)
   x=1;
 }
}
```

У функцији ***draw()*** се проверава да ли има података који се могу прочитати на серијском порту путем кога шаље измерене вредности. Ако има података, чита се вредност до првог знака за нови ред функцијом ***readStringUntil()***. Ова вредност се претвара у број и приказује се на позицији **(x, val)** где је **x** тренутна позиција на којој треба нацртати вредност.

Вредност се приказује тако што се црта једна црна линија на координати **x** од позиције 1 до очитане позиције и једна бела линија на истој координати од позиције до врха (који треба да буде мало мањи од вредности ***height*** како се не би пребрисала вертикална стрелица). Пошто се прикаже ова вредност, координата **x** се повећава за 1 како би се следећа измерена вредност приказала на новој локацији. Ако је после повећавања вредност променљиве **x** већа од жељене ширине на којој желимо да исцртавамо вредности, вредност координате **x** поново постављамо на 1 како би се мерене вредности поново исцртавале од почетка.