Inheritance in PyGame

Lets try to create 2 different types of ball.

1. Fast ball which is hollow from inside
2. Slow ball which is solid

This requires us to rethink the entire strucure of the OOP's approach.
since both the balls have a few things in common

Attributes

1. x and y coordinates
2. radius
3. color

Methods

1. move()

Task 1:

- **So our approach will be to create a base/parent class which has these attributes and then inherit from this class and create child classes.**
- **This will automatically give us the common attributes and then we can do the necessary modifications/addition to get the desired output.**

In [1]:
```
# creating the Base/Parent cirlce class
class Circle():
    def __init__(self):
        self.x = randint(0,500)
        self.y = randint(0,500)
        self.r = randint(10,50)
        self.color = (randint(0,255),randint(0,255),randint(0,255))
        self.x_speed = randint(-2,2)
        self.y_speed = randint(-2,2)

    def move(self):
        self.x = self.x+self.x_speed
        self.y = self.y+self.y_speed
```

Once the base class is created with the attributes common to both the classes we can inherit from

this class and create the child classes.

Task 2 :Creating the FastCircle Class

In [2]:
```
# Inheriting from the parent circle class and creating the child class for FasCtCircle
class FastCircle(Circle):
    def  __init__(self):
        super().__init__()

    # redefinig the move() method
    def move(self):
        self.x = self.x + (self.x_speed*2)
        self.y = self.y + (self.y_speed*2)

    # Adding a new method for drawing circles with thin edges
    def draw(self):
        pygame.draw.circle(screen, self.color, (self.x, self.y), self.r,2)
```

Task 3: Creating the SlowCircle Class

In [2]:
```
# Inheriting from the parent circle class and creating the child class for SlowCircle
class SlowCircle(Circle):
    def  __init__(self):
        super().__init__()

    #no need to define the move method as we can use the one from the parent class

    # Adding a new method for drawing circles filled color
    def draw(self):
        pygame.draw.circle(screen, self.color, (self.x, self.y), self.r)
```

Putting it all together

In [1]:
```
import pygame
from random import randint

pygame.init()

screen = pygame.display.set_mode([500, 500])   # creates a screen with the said size

clock = pygame.time.Clock()


# creating the Base/Parent cirlce class
class Circle():
    def __init__(self):
        self.x = randint(0,500)
        self.y = randint(0,500)
        self.r = randint(10,50)
        self.color = (randint(0,255),randint(0,255),randint(0,255))
        self.x_speed = randint(-2,2)
        self.y_speed = randint(-2,2)

    def move(self):
        self.x = self.x+self.x_speed
        self.y = self.y+self.y_speed
```

```
# Inheriting from the parent circle class and creating the child class for FasCtCircle
class FastCircle(Circle):
    def  __init__(self):
        super().__init__()

    # redefinig the move() method
    def move(self):
        self.x = self.x + (self.x_speed*2)
        self.y = self.y + (self.y_speed*2)

    # drawing circles with thin edges
    def draw(self):
        pygame.draw.circle(screen, self.color, (self.x, self.y), self.r,2)


# Inheriting from the parent circle class and creating the child class for SlowCircle
class SlowCircle(Circle):
    def  __init__(self):
        super().__init__()

    #no need to define the move method as we can use the one from the parent class

    # drawing circles with thin edges
    def draw(self):
        pygame.draw.circle(screen, self.color, (self.x, self.y), self.r)



# creating circle objects using the FastCircle class
c1 = FastCircle()
c2 = FastCircle()
c3 = FastCircle()

# creating circle objects using the SlowCircle class
c4 = SlowCircle()
c5 = SlowCircle()
c6 = SlowCircle()


run = True
while run:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            run = False

    screen.fill((255,255,255))



    # creating individual cirlce objects using the draw method
    # creating Fastcircles
    c1.draw()
    c2.draw()
    c3.draw()
    # Creating SlowCircles
    c4.draw()
    c5.draw()
    c6.draw()
```

```
    # Moving the circle objects
    # Moving FastCircles
    c1.move()
    c2.move()
    c3.move()
    # Moving SlowCircles
    c4.move()
    c5.move()
    c6.move()



    pygame.display.flip()

    clock.tick(30)


pygame.quit()
```

```
pygame 2.0.1 (SDL 2.0.14, Python 3.6.8)
Hello from the pygame community. https://www.pygame.org/contribute.html
```

->Adding OOP's does help up in keeping all the attributes and methods associated with an object in a single place.

-> It also helps us reduce the code size a lot by reusing the code by inheriting.

-> But still we have to call the methods on individual objects which is not so efficient

Instead of assigning each objects a name, we can directly add the items into the list and call individual items by indexing them.

In [1]:

```
import pygame
from random import randint

pygame.init()

screen = pygame.display.set_mode([500, 500])   # creates a screen with the said size

clock = pygame.time.Clock()


# creating the Base/Parent cirlce class
class Circle():
    def __init__(self):
        self.x = randint(0,500)
        self.y = randint(0,500)
        self.r = randint(10,50)
        self.color = (randint(0,255),randint(0,255),randint(0,255))
        self.x_speed = randint(-2,2)
        self.y_speed = randint(-2,2)

    def move(self):
        self.x = self.x+self.x_speed
        self.y = self.y+self.y_speed
```

```
# Inheriting from the parent circle class and creating the child class for FasCtCircle
class FastCircle(Circle):
    def  __init__(self):
        super().__init__()

    # redefinig the move() method
    def move(self):
        self.x = self.x + (self.x_speed*2)
        self.y = self.y + (self.y_speed*2)

    # drawing circles with thin edges
    def draw(self):
        pygame.draw.circle(screen, self.color, (self.x, self.y), self.r,2)


# Inheriting from the parent circle class and creating the child class for SlowCircle
class SlowCircle(Circle):
    def  __init__(self):
        super().__init__()

    #no need to define the move method as we can use the one from the parent class

    # drawing circles with thin edges
    def draw(self):
        pygame.draw.circle(screen, self.color, (self.x, self.y), self.r)



# creating circle objects using the FastCircle class
cir =[]

for i in range(5):
    cir.append(FastCircle())

for i in range(5):
    cir.append(SlowCircle())

run = True
while run:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            run = False

    screen.fill((255,255,255))



    # creating individual cirlce objects using the draw method
    # creating Fastcircles
    cir[1].draw()
#     c2.draw()
#     c3.draw()
#     # Creating SlowCircles
#     c4.draw()
#     c5.draw()
#     c6.draw()

#     # Moving the circle objects
#     # Moving FastCircles
#     c1.move()
#     c2.move()
```

```
#     c3.move()
#     # Moving SlowCircles
#     c4.move()
#     c5.move()
#     c6.move()



    pygame.display.flip()

    clock.tick(30)


pygame.quit()
```

```
pygame 2.0.1 (SDL 2.0.14, Python 3.6.8)
Hello from the pygame community. https://www.pygame.org/contribute.html
```

Polymorphism

As we have objects from 2 differnet classes and both of them have methods with the same name doing different things we can take advantage of polymorphism to get all the methods called uinsg a single for loop.

In [1]:

```
import pygame
from random import randint

pygame.init()

screen = pygame.display.set_mode([500, 500])   # creates a screen with the said size

clock = pygame.time.Clock()


# creating the Base/Parent cirlce class
class Circle():
    def __init__(self):
        self.x = randint(0,500)
        self.y = randint(0,500)
        self.r = randint(10,50)
        self.color = (randint(0,255),randint(0,255),randint(0,255))
        self.x_speed = randint(-2,2)
        self.y_speed = randint(-2,2)

    def move(self):
        self.x = self.x+self.x_speed
        self.y = self.y+self.y_speed


# Inheriting from the parent circle class and creating the child class for FasCtCircle
class FastCircle(Circle):
    def  __init__(self):
        super().__init__()

    # redefinig the move() method
    def move(self):
        self.x = self.x + (self.x_speed*2)
        self.y = self.y + (self.y_speed*2)
```

```
    # drawing circles with thin edges
    def draw(self):
        pygame.draw.circle(screen, self.color, (self.x, self.y), self.r,2)


# Inheriting from the parent circle class and creating the child class for SlowCircle
class SlowCircle(Circle):
    def  __init__(self):
        super().__init__()

    #no need to define the move method as we can use the one from the parent class

    # drawing circles with thin edges
    def draw(self):
        pygame.draw.circle(screen, self.color, (self.x, self.y), self.r)

# List to store all the circles
cir = []

# creating FastCircles and adding them to the list
for i in range(5):
    cir.append(FastCircle())

# creating SlowCircles and adding them to the list
for i in range(5):
    cir.append(SlowCircle())


run = True
while run:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            run = False

    screen.fill((255,255,255))



    # creating individual cirlce objects using the draw method
    for i in range(5):
        cir[i].draw()

    # Moving the circle objects
    for i in range(5):
        cir[i].move()



    pygame.display.flip()

    clock.tick(30)


pygame.quit()
```

```
pygame 2.0.1 (SDL 2.0.14, Python 3.9.1)
Hello from the pygame community. https://www.pygame.org/contribute.html
```

HOMEWORK

1. Make changes to the final code so that there is no circle which is stationary.
   Note : Go through the documentation of the Random module and try to find a function which will help with the above task.
   https://docs.python.org/3/library/random.html

In [ ]: