Unit 2: Using Objects
Objects: Instances of Classes

Adapted from:

1) Building Java Programs: A Back to Basics Approach by Stuart Reges and Marty Stepp

2) Runestone CSAwesome Curriculum

https://longbaonguyen.github.io

Class

In this unit, you will learn to use **objects** (variables of a **class or reference type**) that have been designed by other programmers.

Later on, in Unit 5, you will learn to create your own classes and objects.

A **class** in programming defines a new abstract data type. A class is the formal implementation, or blueprint, of the attributes and behaviors of an object.

When you create **objects or instances of a class** in coding, you create new variables or objects of that class data type.

Objects

class: A program entity that represents a template for a new type of objects.

object or instance: An entity that combines **attributes(data)** and **behavior(methods)**.

- **object-oriented programming (OOP)**: Programs that perform their behavior as interactions between objects. Java is object-oriented.

The `Car` class is a template for creating `Car` objects.

Classes and objects

Example:

The Ipod **class** provides the template or blueprint for the **attributes(data)** and **behavior(methods)** of an ipod **object.**

Its attributes or data can include the current song, current volume and battery life.

Its behavior or methods can include change song, change volume, turn on/off, etc…

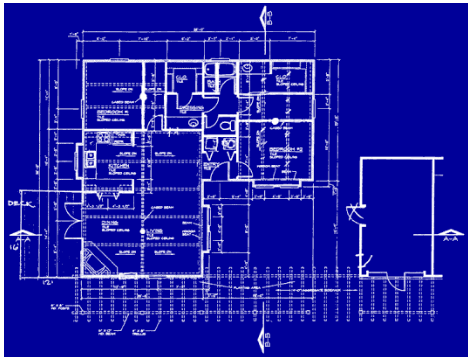
Two different Ipod objects can have different attributes/data. However, their template share the same implementation/code.

Blueprint analogy

iPod blueprint

attributes:
- current song
- volume
- battery life

behavior:
- power on/off
- change station/song
- change volume
- choose random song



creates

iPod #1

attributes:
- song = "Uptown Funk"
- volume = 17
- battery life = 2.5 hrs

behavior:
- power on/off
- change station/song
- change volume
- choose random song

iPod #2

attributes:
- song = "You make me wanna"
- volume = 9
- battery life = 3.41 hrs

behavior:
- power on/off
- change station/song
- change volume
- choose random song

iPod #3

attributes:
- song = "Trumpets"
- volume = 24
- battery life = 1.8 hrs

behavior:
- power on/off
- change station/song
- change volume
- choose random song

More Examples

Suppose you are writing an arcade game. What are some useful classes and their corresponding objects?

Example:

The **Character** Class represents characters in the game.

Attributes/Data: String name, int numberOfLives, boolean isAlien.

Behavior/Methods: shoot(), runLeft(), runRight(), jump().

Objects:

```
Character player1, player2; //declaring objects of type Character
Character enemy1, enemy2;
```

More Examples

Your game might have more than one classes.

Classes: Character, Boss, MysteryBox, Obstacle.

Objects:

```
Boss level1, level2;
MysteryBox yellow; // give player 3 extra lives
MysteryBox red; // give player 100 coins
Obstacle wall; //immovable
Obstacle poison; // kills player
```

Sprite

Soon, we will use the Processing IDE(www.processing.org) to help us write arcade games.

In games, a **sprite** is an object that represents a character in a game.

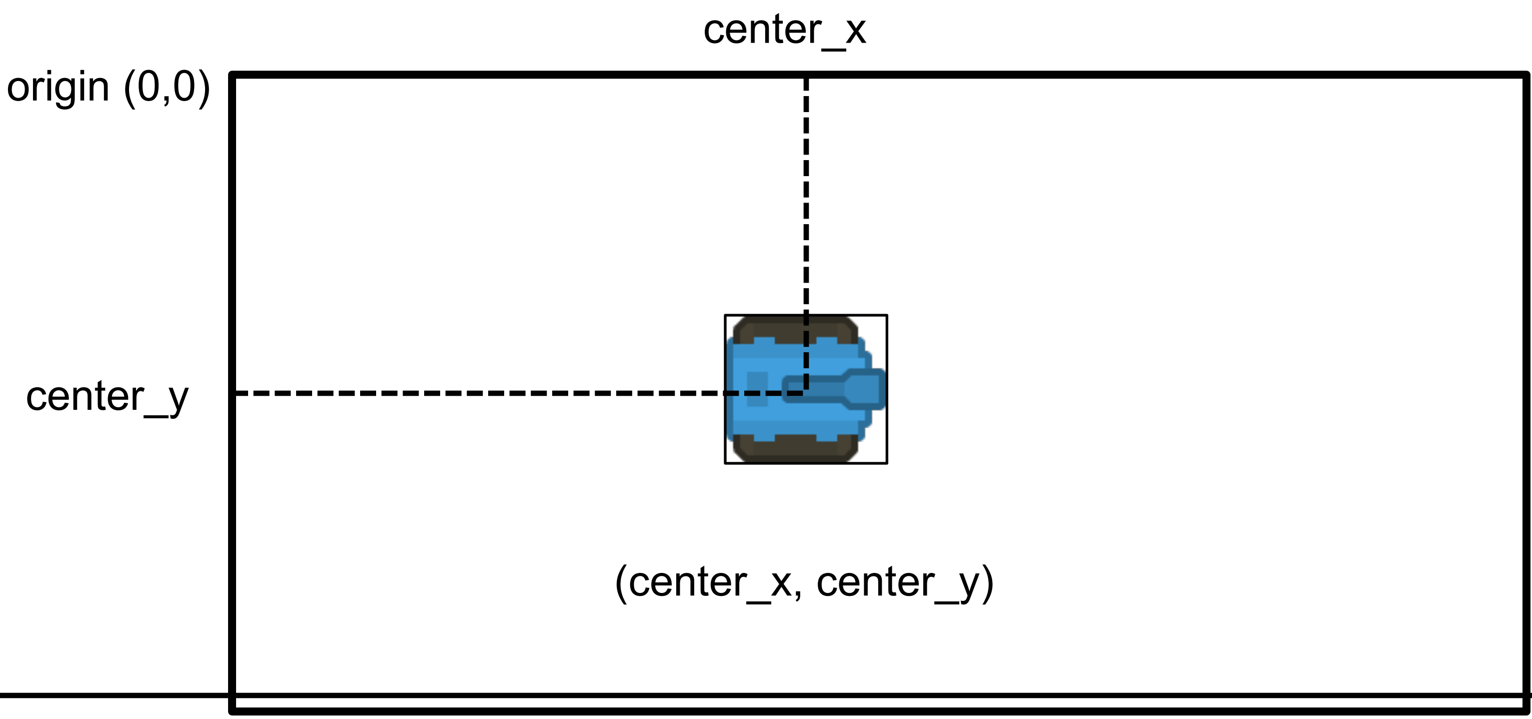
It usually consists of an image(or set of images) of a character moving and interacting with other sprites in the game.

Let's look at an example of a Sprite class that we will expand later into a full arcade game.

Sprite

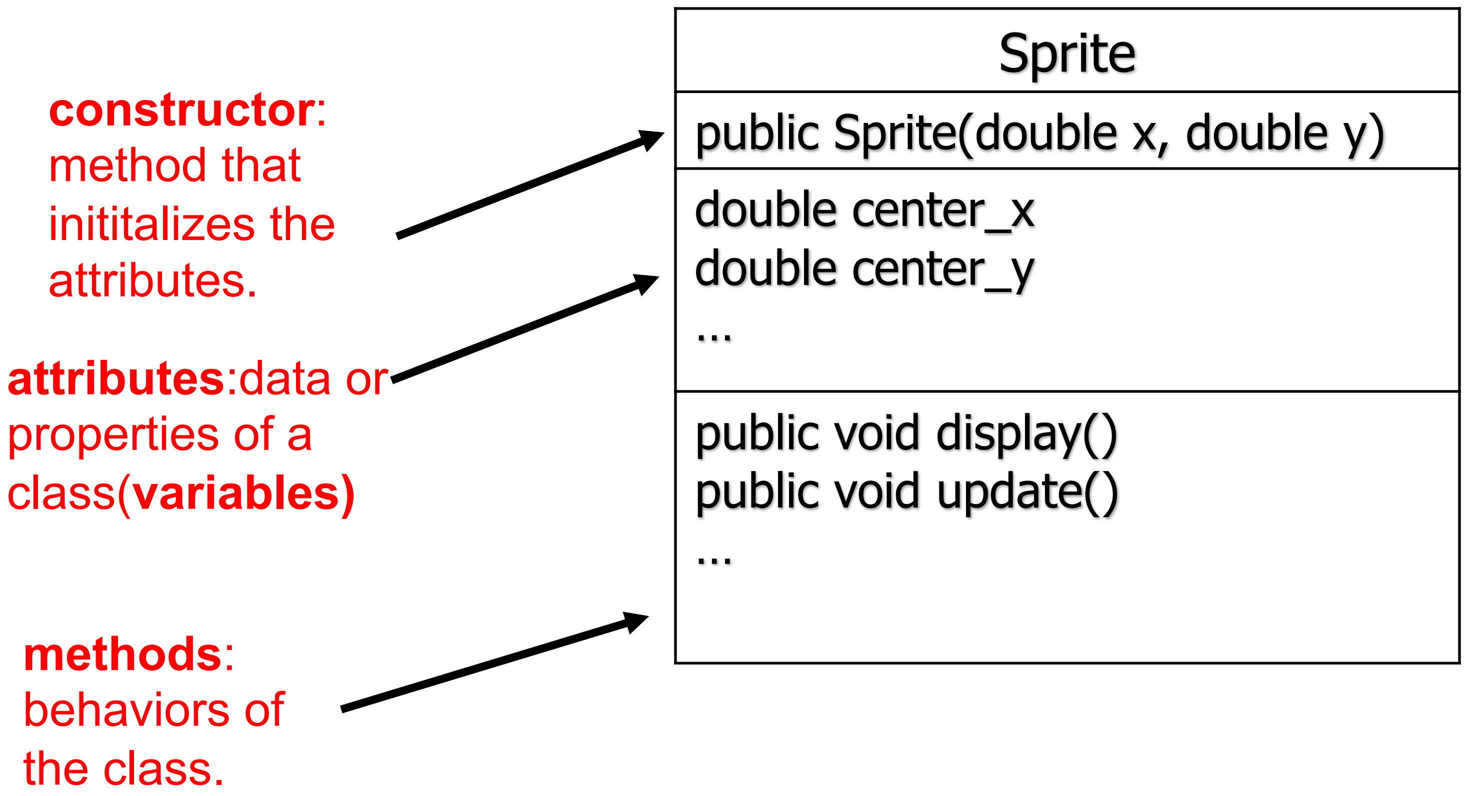
Sprite's attributes can include many properties: the image(.png or .jpg) of the sprite, the width and height of the image, and position on the screen given by center_x and center_y instance variables.

To keep things simple, for now, we focus on just two attributes: center_x and center_y.



Class Diagram

The image below represents a **class diagram** of the Sprite class. The class diagram allows us to preview the contents of the class.



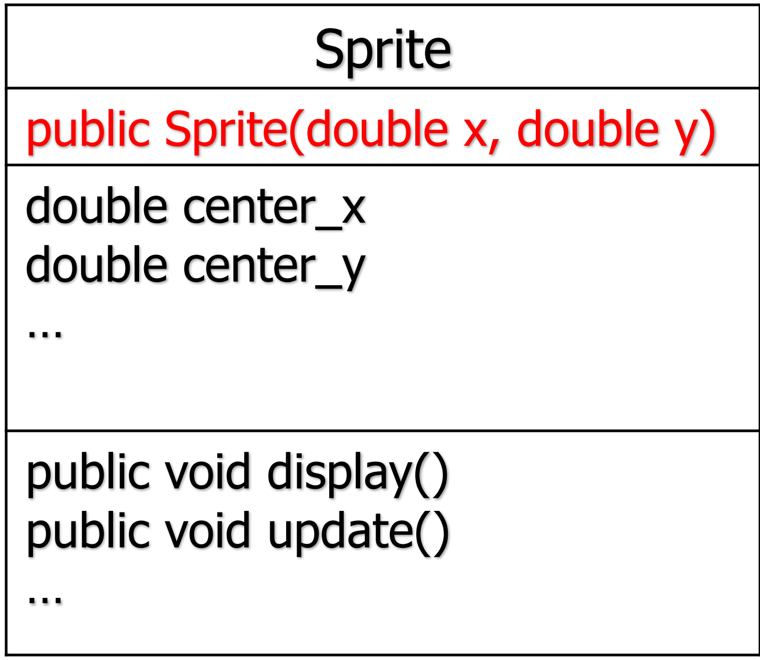
constructor: method that inititalizes the attributes.
attributes:data or properties of a class(**variables)**
methods: behaviors of the class.
Sprite
public Sprite(double x, double y)
double center_x
double center_y
…
public void display()
public void update()
…

Constructor

The **constructor** of a class is a method that allows us to initialize the attributes(variables) of an object when it is first created.

Constructors always have the same name as the class and are used with the keyword **new**.

signature: name of constructor and its parameter list.



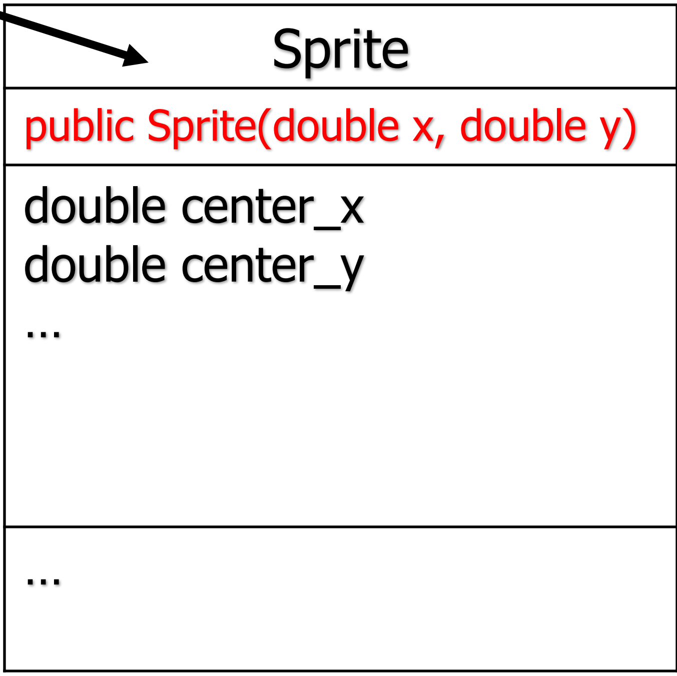
An object variable is created using the keyword new followed by a call to a constructor.

Constructor

Consider the line of code used to create a Sprite object called player:

```
Sprite player = new Sprite(30.0,50.0);
```

arguments or parameters: data that methods need to do its job.



The **actual parameters** (30.0, 50.0) is passed to the **formal parameters** (double x, double y) of the constructors. The actual parameters passed to a constructor must be compatible with the types identified in the formal parameter list.

In code not shown here, the variables x and y are then used to initialize the attributes center_x and center_y.

Multiple Objects

We can create multiple objects using the constructor.

```
public class ConstructorExample
{
  public static void main(String[] args){
       Sprite player1 = new Sprite(30, 50);
       Sprite player2 = new Sprite(10, 40);
  }
}
```

Note that in this example, player1 and player2 are two different objects or instances of the same class, each with its own copy of instance variables and methods.

We can access the attributes of an object by using the **dot notation** as shown in the next example.

Accessing attributes

We can **access** the attributes of an object by using the **dot notation.**

```
public class ConstructorExample{
  public static void main(String[] args){
        Sprite player1 = new Sprite(30, 50);
        Sprite player2 = new Sprite(10, 40);
        System.out.println(player1.center_x) // 30.0
        System.out.println(player1.center_y) // 50.0
        System.out.println(player2.center_x) // 10.0
        System.out.println(player2.center_y) // 40.0


  }
}
```

modifying attributes

We can **modify** the attributes of an object by using the **dot notation.**

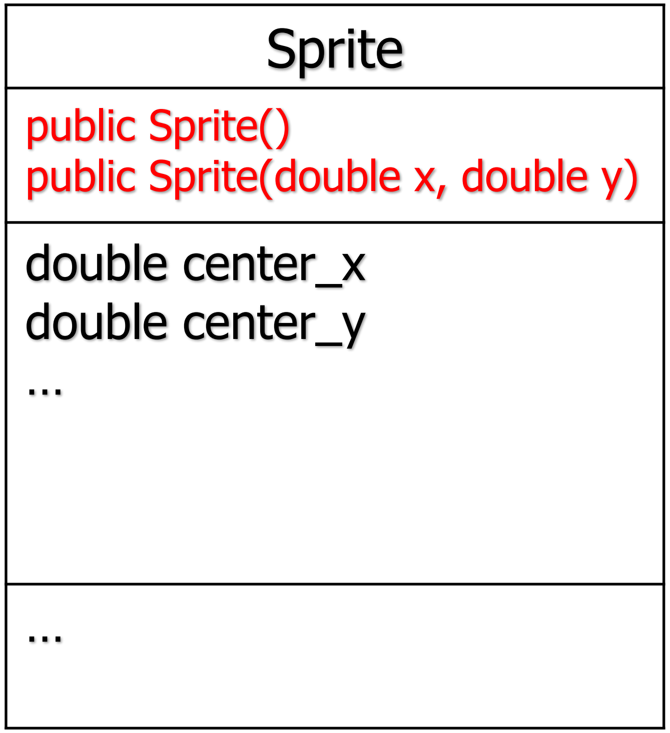
```
public class ConstructorExample{
  public static void main(String[] args){
       Sprite player1 = new Sprite(30, 50);
       Sprite player2 = new Sprite(10, 40);
       System.out.println(player1.center_x) // 30.0
       System.out.println(player1.center_y) // 50.0
       System.out.println(player2.center_x) // 10.0
       System.out.println(player2.center_y) // 40.0
       player1.center_x = 100;
       System.out.println(player1.center_x) // 100.0


  }
```

Overloaded constructors

Constructors are said to be **overloaded** when there are multiple constructors with the same name but a different signature.

Note on the right, the Sprite class has two constructors: one that has no parameter and one that has parameters.

Usually, the constructor that has no parameter (sometimes called the default constructor) initializes the object to some default values, for example, zeroes.



```
Sprite
public Sprite()
public Sprite(double x, double y)
double center_x
double center_y
…
…
```

overloaded constructors

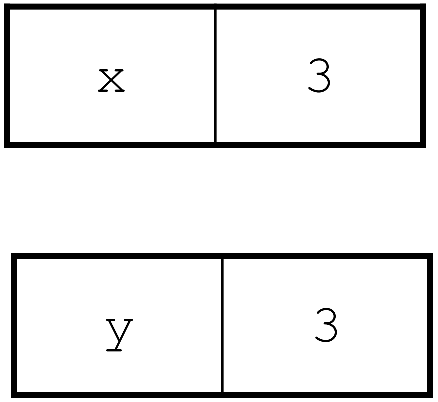
We can call different constructors to initialize our objects. **Assume the default constructor initializes center_x and center_y to be at the origin.**

```
public class ConstructorExample{
  public static void main(String[] args){
        Sprite player1 = new Sprite();
        Sprite player2 = new Sprite(10, 40);
        System.out.println(player1.center_x) // 0.0
        System.out.println(player1.center_y) // 0.0
        System.out.println(player2.center_x) // 10.0
        System.out.println(player2.center_y) // 40.0


  }
}
```

Primitive vs. Reference Type

The memory associated with a variable of a **primitive type(int, double, boolean)** holds an actual primitive value.

```
int x = 3; // x is a variable of a primitive type
           // the memory associated with x actually holds the value 3

int y = x; // y copies the value of x
           // y is a different variable in memory
           // which also hold the value 3
```

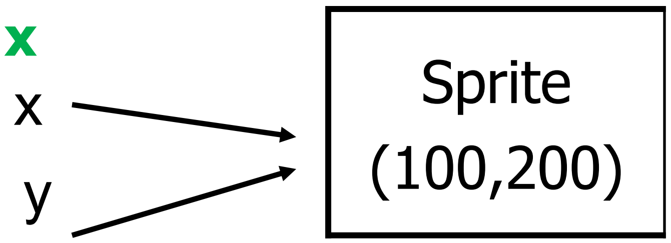


Here we have **two different integers** in memory
both of which has the value 3.

Primitive vs. Reference Type

While the memory associated with a variable of a **reference type** holds an object **reference value**. This value is the **memory address** of the referenced object.

```
Sprite x = new Sprite(100, 200);
// x is a variable of a reference type
// the value of x is actually an address in memory of this
// Sprite object not the actual object itself.
Sprite y = x; // copies the address of x
```



Note that this is similar to the previous slide example. But in this case, both x and y stores the **same address in memory** therefore both refer to the **same object.**

More Examples

```
// x, y and isPrime are variables of primitive type(Unit 1).
int x = 3;
double y = 2.5;
boolean isPrime = false;

// player1, player2 are variables of a reference type

Sprite player1 = new Sprite(100, 200);
Sprite player2 = new Sprite();
```

Null

The keyword **null** is a special value used to indicate that a reference is not associated with any object. Accessing an instance variable of a null reference will result in a NullPointerException.

```
// player1 points to the address or location in memory for the
// Sprite object
Sprite player1 = new Sprite(100, 200);
Sprite player2 = null; // player2 is initialized to null since it is not
                       // yet associated with any object.
System.out.println(player2.center_x)  // NullPointerException
Sprite player3;
System.out.println(player3.center_x)  // error! Player3 is not
                                      //  initialized(not a NullPointerException)
```

Example Implementation

Although we will write the Sprite class later in Unit 5. It is instructive to see the implementation of this very simple class to understand the structure of a class.

Note that there are two .java files. The main method is in Main.java(sometimes call the driver class) and the Sprite object class is in Sprite.java.

Main.java

```
public class Main{
  public static void main(String[] args){
      Sprite player1 = new Sprite();
      Sprite player2 = new Sprite(10, 40);
  }
}
```

creating the objects by calling one of the constructors

Sprite.java

declaring the instance variables

```
public class Sprite{
  double center_x;
  double center_y;
  public Sprite(){
     center_x = 0;
     center_y = 0;
  }
  public Sprite(double x, double y){
     center_x = x;
     center_y = y;
  }
}
```

overloaded constructors

initializing the instance variables

References

1) Building Java Programs: A Back to Basics Approach by Stuart Reges and Marty Stepp

2) Runestone CSAwesome Curriculum: https://runestone.academy/runestone/books/published/csawesome/index.html

For more tutorials/lecture notes in Java, Python, game programming, artificial intelligence with neural networks:

https://longbaonguyen.github.io