Java

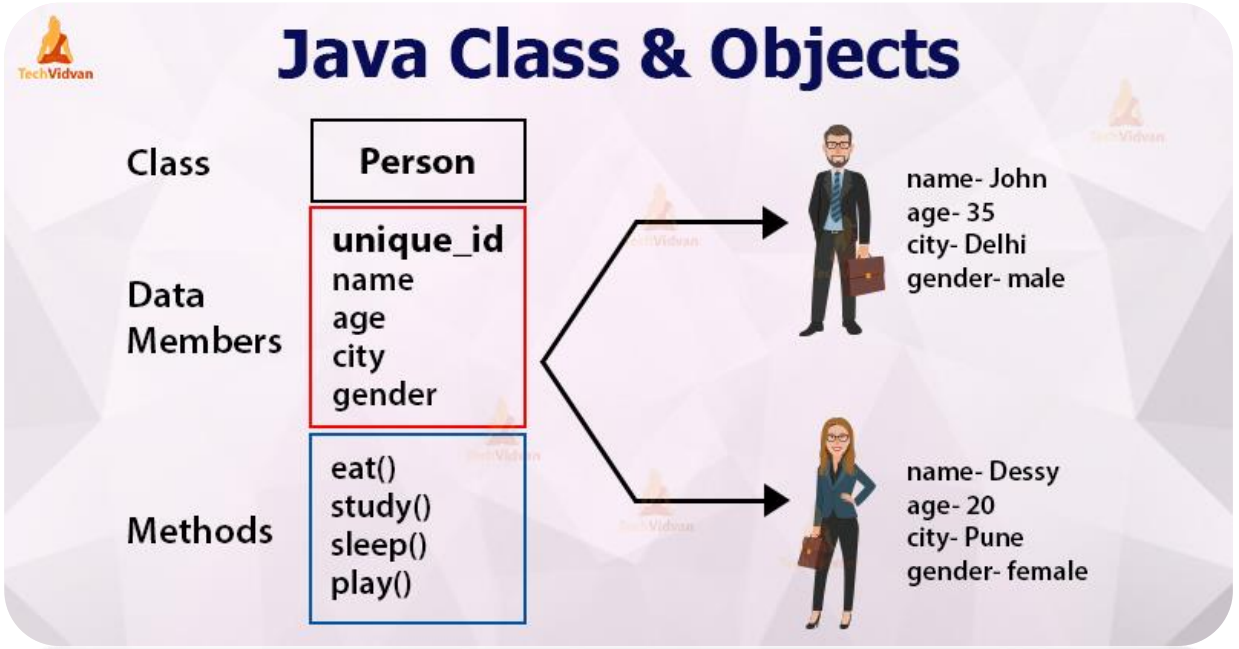
Language constructs for introduction to Object Oriented Programming

What is an object?

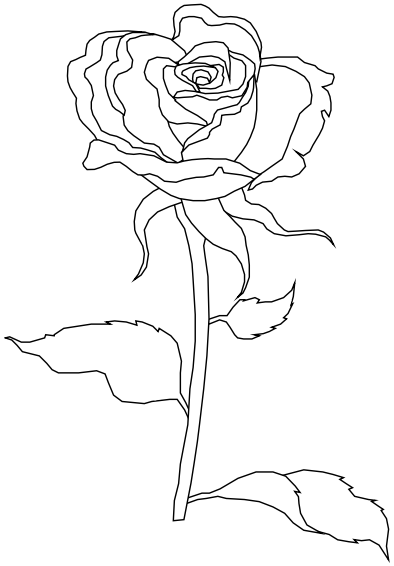
- Java supports the view that programs are composed of objects that interact with one another.
- How would you describe an object?
- Using its
  - characteristics (has a ----?) and
  - its behaviors (can do ----?)
- Object must have unique identity (name) : Person
- Consider a person:
  - Name and Birthdate are characteristics (Data Declarations)
  - walk, speak, sit are behaviors (Methods)

Defining classes for objects

- A class defines the properties and behaviors for objects.
- An *object* represents an entity in the real world that can be distinctly identified.
- For example, a student, a desk, a circle, a button, and even a loan can all be viewed as objects.



Example



class Rose

objects

class

Object References

redRose

blueRose

Defining classes for objects

- An object has a unique identity, state, and behaviour.

  - The *state* of an object (*properties* or *attributes*) is represented by *data fields* with their current values.
  - A circle object, has a data field **radius**, which is the property that characterizes a circle.

  - The behavior of an object (actions) is defined by methods.
  - You may define methods named getArea() and getPerimeter() for circle objects.
    - A circle object may invoke getArea() to return its area and getPerimeter() to return its perimeter.
  - You may also define the setRadius(radius) method.
    - A circle object can invoke this method to change its radius.

Defining classes for objects

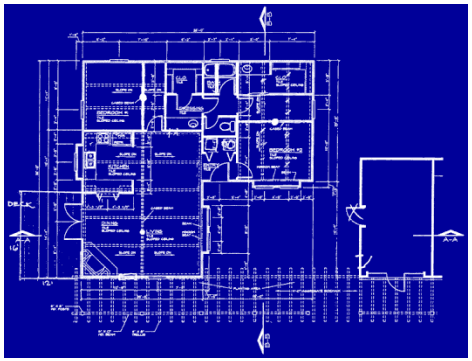
- A *class* is a template, blueprint, or *contract* that defines what an object's data fields and methods will be.
- An object is an instance of a class.
  - You can create many instances of a class.
  - Creating an instance is referred to as *instantiation.*
- A Java class uses variables to define data fields and methods to define actions/behaviour.
- Additionally, a class provides methods of a special type, known as *constructors*, which are invoked to create a new object.
- A constructor can perform any action, but constructors are designed to perform initializing actions, such as initializing the data fields of objects.

Blueprint analogy

iPod blueprint

state:
- current song
- volume
- battery life

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



creates

iPod #1

state:
- song = "1,000,000 Miles"
- volume = 17
- battery life = 2.5 hrs

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

iPod #2

state:
- song = "Letting You"
- volume = 9
- battery life = 3.41 hrs

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

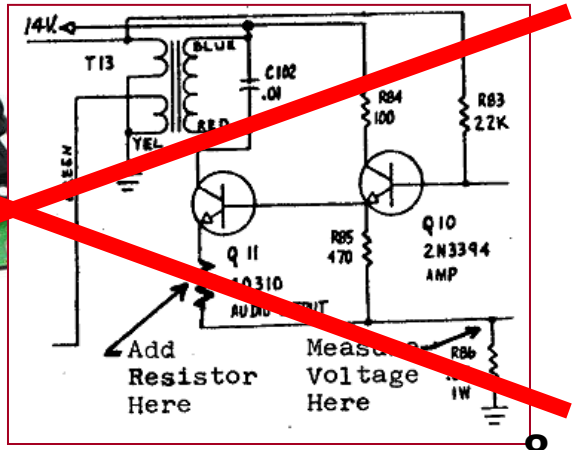
iPod #3

state:
- song = "Discipline"
- volume = 24
- battery life = 1.8 hrs

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

Abstraction

- **abstraction**: A distancing between ideas and details.
  - We can use objects without knowing how they work.
- abstraction in an iPhone:
  - You understand its external behavior (buttons, screen).
  - You may not understand its inner details,
    and you don't need to if you just want to use it.



The `String` Class

- *A* **String** *object is immutable: Its content cannot be changed once the string is created.*
- Constructing a String:

  String newString = **new** String(stringLiteral);

  Examples:

```
String message = new String("Welcome to Java");
String s = new String();
// Java treats a string literal as a String object.
String message = "Welcome to Java";
```

- You can also create a string from an array of characters. For example, the following statements create the string **"Good Day"**:

```
char[] charArray = {'G', 'o', 'o', 'd', ' ', 'D', 'a', 'y'};
String message = new String(charArray);
```

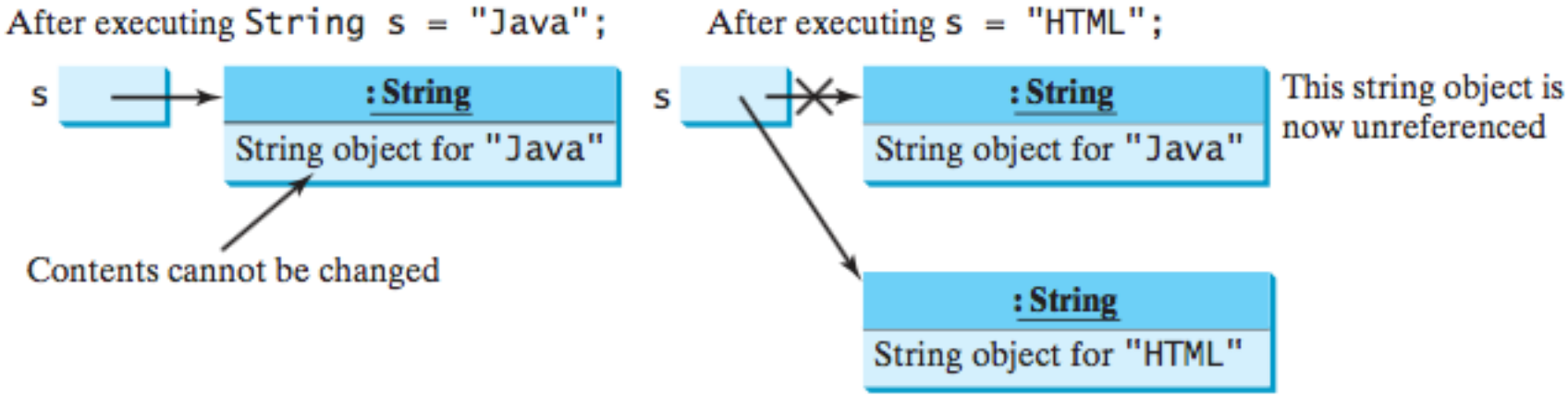
The `String` Class

A **String** object is immutable; its contents cannot be changed. Does the following code change the contents of the string?

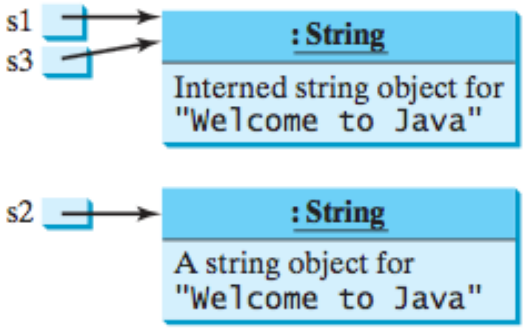
```
String s = "Java";
s = "HTML";
```

- The answer is no. The first statement creates a **String** object with the content **"Java"** and assigns its reference to **s**.
- The second statement creates a new **String** object with the content **"HTML"** and assigns its reference to **s**.
- The first **String** object still exists after the assignment, but it can no longer be accessed, because variable **s** now points to the new object, as shown below:

The `String` Class




```
String s1 = "Welcome to Java";

String s2 = new String("Welcome to Java");

String s3 = "Welcome to Java";

System.out.println("s1 == s2 is " + (s1 == s2));
System.out.println("s1 == s3 is " + (s1 == s3));
```



display

```
s1 == s2 is false
s1 == s3 is true
```

s1 == s2 is **false**, because **s1** and **s2** are two different string objects, even though they have the same contents.

`Point` objects

```
Point p1 = new Point(5, -2);
Point p2 = new Point(); // origin, (0, 0)
```

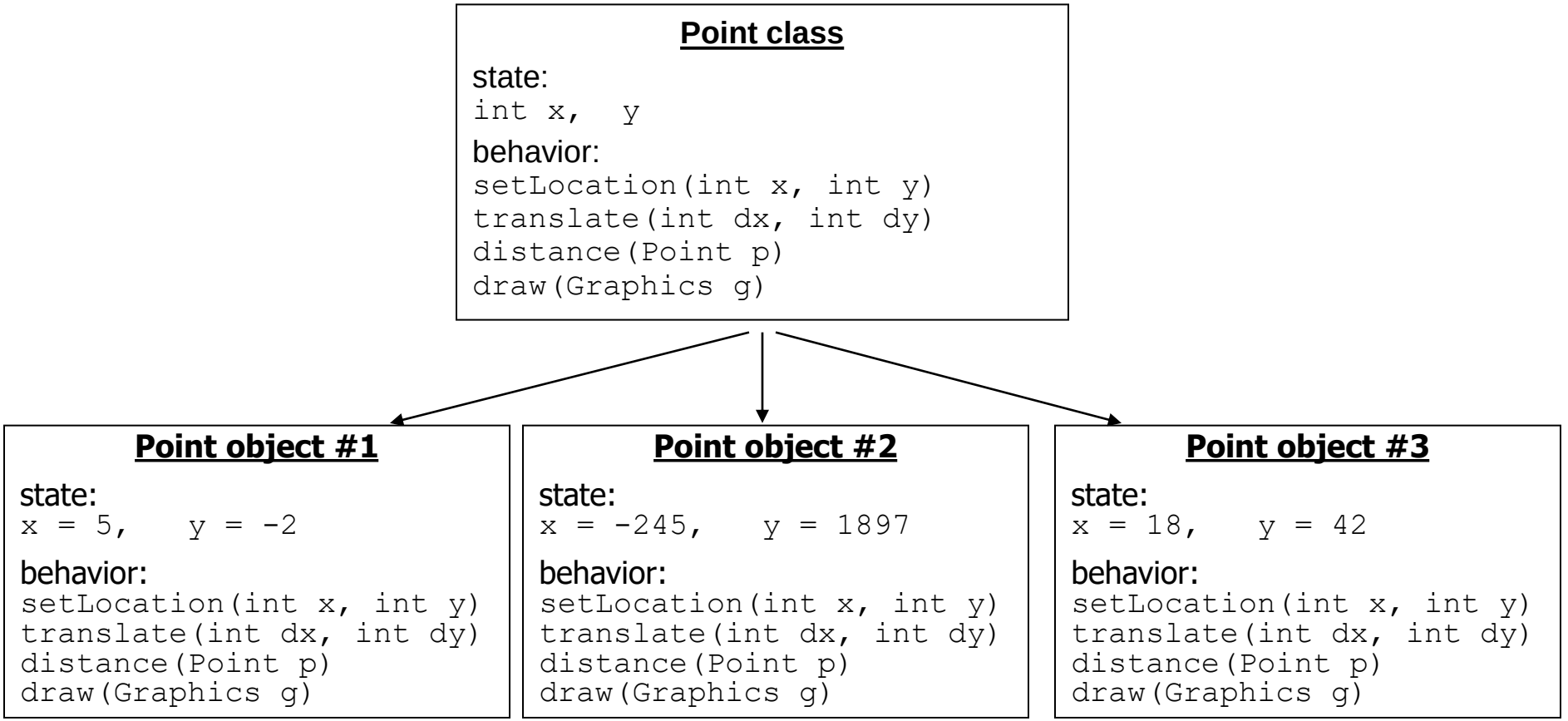
- Data in each `Point` object:

Field name	Description
`x`	the point's x-coordinate
`y`	the point's y-coordinate

- Methods in each `Point` object:

Method name	Description
`toString()`	returns a String representation of this Point
`setColor(Color c)`	Set this Point's color
`distance(Point` **`p`**`)`	how far away the point is from point *p*
`draw(Graphics` **`g`**`)`	displays the point on a drawing panel

`Point` class as blueprint



- The class (blueprint) will describe how to create objects.
- Each object will contain its own data and methods.

Object state: Fields

Object State

- Similar to how two variables can contain different data.
- Attributes: Data that describes each instance or example of a class.
- Different objects have the same attributes, but the values of those attributes can vary
  - The class definition specifies the attributes and methods for *all objects*
- The current value of an object's attributes determines its state.

Age: 35
Weight: 192

Age: 50
Weight: 125

Age: 0.5
Weight: 7

Point class

```
public class Point {
    private int x;
    private int y;
}
```

- Save this code into a file named `Point.java`.

- The above code creates a new type named `Point`.
  - Each `Point` object contains two pieces of data:
    - an `int` named `x`, and
    - an `int` named `y`.
  - `Point` objects do not contain any behavior (yet).

Fields

- **field**: A variable inside an object that is part of its state.
  - Each object has *its own copy* of each field.

- Declaration syntax:

  access_modifier type name;

  - Example:

```
public class Student {
    // each Student object has a name and
    // gpa field (instance variable)
    private String name;
    private double gpa;
}
```

Accessing fields

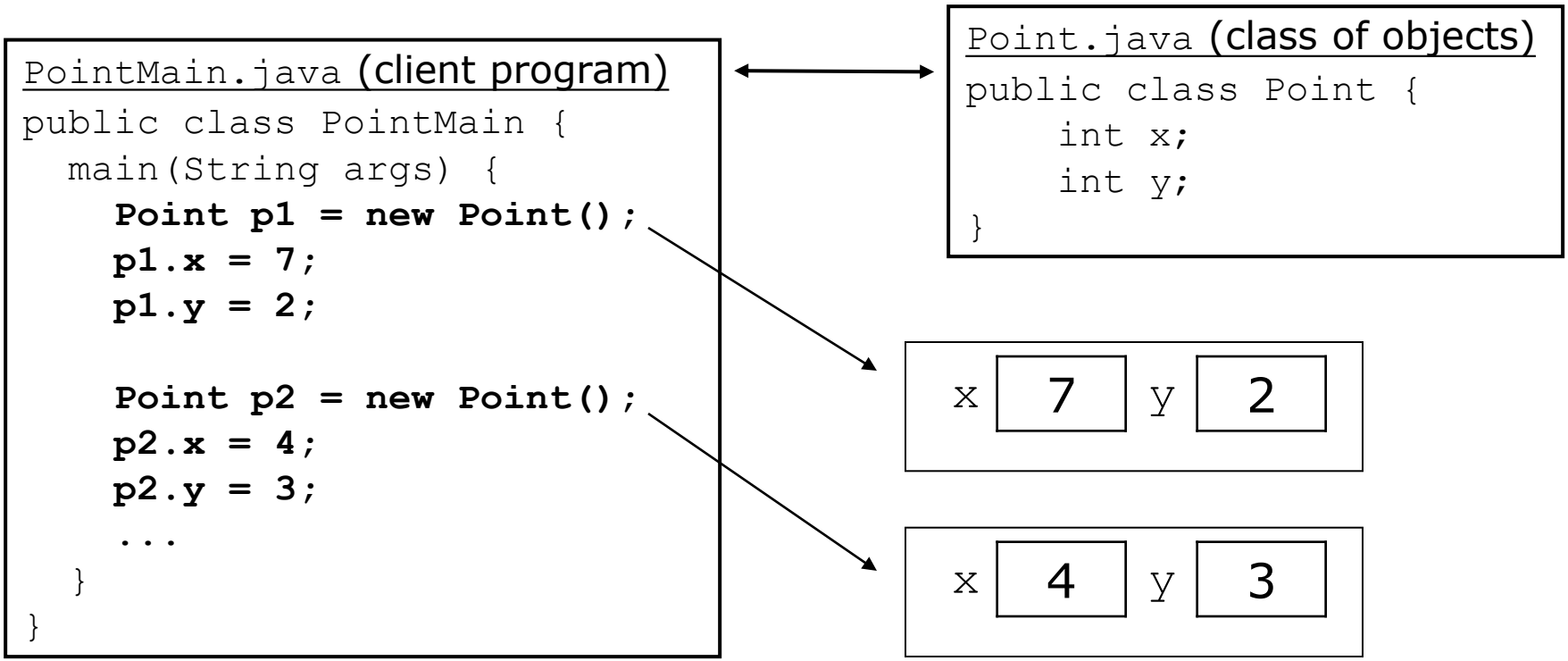
- Other classes can access/modify an object's fields.
  - *depending on the access modifier*
  - access: **variable . field**
  - modify: **variable . field = value;**

- Example:

```
Point p1 = new Point();
Point p2 = new Point();
System.out.println("the x-coord is " + p1.x);    // access
p2.y = 13;                                       // modify
```

A class and its client

- `Point.java` is not, by itself, a runnable program.
  - A class can be used by **client** programs.



Object behavior: Methods

What Are Methods

- Possible behaviors or actions for each instance (example) of a class.

Fly()

Walk()
Talk()

Walk()
Talk()

Swim()

Instance methods

- **instance method** (or **object method**): Exists inside each object of a class and gives behavior to each object.

```
public type name(parameters) {
    statements;
}
```

  - same syntax as static methods, but without `static` keyword

  Example:

```
public void shout() {
    System.out.println("HELLO THERE!");
}
```

Instance method example

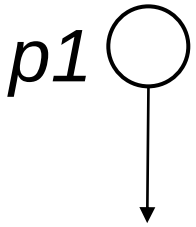
```
public class Point {
    private int x;
    private int y;

    // Draws this Point object with the given pen
    public void draw(Graphics g) {
        ...
    }
}
```

- How will the method know which point to draw?
  - How will the method access that point's x/y data?

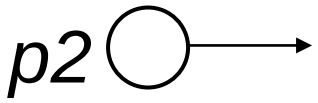
`Point` objects / method

- Each `Point` object has its own copy of the `draw` method, which operates on that object's state:

```
Point p1 = new Point(7, 2);

Point p2 = new Point(4, 3);

p1.draw(g);
p2.draw(g);
```




```
x 7  y 2

public void draw(Graphics g) {
    // this code can see p1's x and y
}
```




```
x 4  y 3

public void draw(Graphics g) {
    // this code can see p2's x and y
}
```

The implicit parameter

- **implicit parameter**:
  The object on which an instance method is called.
  - During the call `p1.draw(g);`
    the object referred to by `p1` is the implicit parameter.
  - During the call `p2.draw(g);`
    the object referred to by `p2` is the implicit parameter.
  - The instance method can refer to that object's fields.
    - We say that it executes in the *context* of a particular object.
    - `draw` can refer to the `x` and `y` of the object it was called on.

Object initialization: Constructors

Constructor

- A constructor is invoked to create an object using the **new** operator.
- Constructors are a special kind of method. They have three peculiarities:
  - A constructor must have the same name as the class itself.
  - Constructors do not have a return type - not even **void**.
  - Constructors are invoked using the **new** operator when an object is created. Constructors play the role of initializing objects.

Constructor

- A class normally provides a constructor without arguments (e.g., **Circle()**). Such a constructor is referred to as a *no-arg* or *no-argument constructor*.
- A class may be defined without constructors. In this case, a public no-arg constructor with an empty body is implicitly defined in the class.
  - This constructor, called a *default constructor*, is provided automatically *only if no constructors are explicitly defined in the class*.

Constructor

- Different versions of the constructor can be implemented with different initializations e.g., one version sets all attributes to default values while another version sets some attributes to the value of parameters.

```
public Person(int anAge) {
    age = anAge;
    name = "No-name";
 }
```

```
public Person() {
  age = 0;
  name = "No-name";
}
```

```
// Calling the versions (distinguished by parameter list)
Person p1 = new Person(100);
```

```
Person p2 = new Person();
```


JAVA