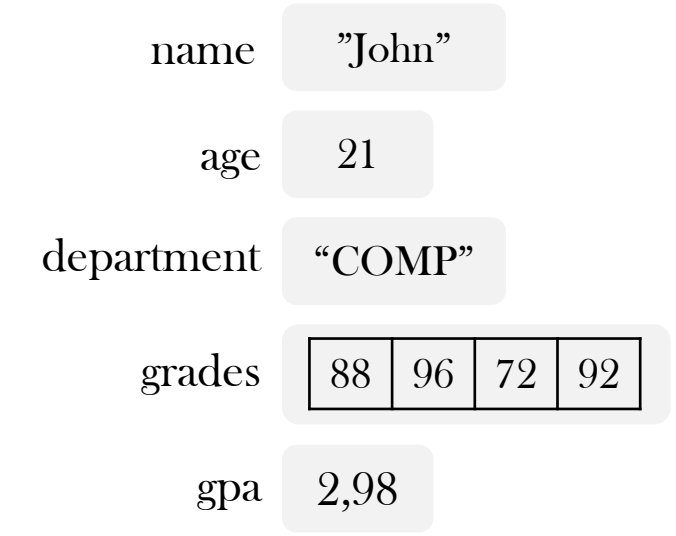
Classes and Objects

What are objects?

- Spotify: Songs, Albums, Artists
- Games: Characters, Maps
- Netflix: Movies, Actors, My Movie List
- Instagram: Users, Posts, Photos

How to model a student?

- Students have
  - Name: “John” (Data type: String)
  - Age: 21 (Data type: int)
  - Department: ”COMP” (Data type: String)
  - Grades: {88, 96, 72, 92} (Data type: int[])
  - GPA: 2.98 (Data Type: double)



- In classical programming approach, we can model a student using variables

```
String name = “John”;            // student name
int age = 21;                    // age
String department = “COMP”;      // department
int[] grades = new grades[4];    // array initially contains zeros
double gpa = 2.98;               // student gpa
```

How to model a student?

- Source code for student program
- Let's write printStudent method which displays all student information

```
public static void main(String[] args) {

 String name = "John";       // student name
 int age = 21;               // age
 String department = "COMP";  // department
 int[] grades = new int[10];  // array initially contains zeros
 double gpa = 2.98;          // student gpa

 // printStudent method displays student information
 printStudent(name, age, department, grades, gpa);
}
```

How to model a student?

- printStudent method for students

```
/**
 * Prints student information
 * @param name name
 * @param age age
 * @param department department
 * @param grades grades
 * @param gpa gpa
 */
private static void printStudent(String name, int age, String department, int[] grades, double gpa) {
  System.out.println("\nName       : " + name);
  System.out.println("Age        : " + age);
  System.out.println("Department : " + department);
  System.out.println("GPA        : " + gpa);
  System.out.print("Grades     : ");
  for (int e : grades)
    if (e == -1)
      break;
    else
      System.out.print(e + ", ");
  System.out.print("\n");
}
```

How to model a student?

- Let's have two students: John and Mary

```
public static void main(String[] args) {

 String name1 = "John";
 int age1 = 21;
 String department1 = "COMP";

 String name2 = ”Mary";
 int age2 = 20;
 String department2 = ”EE";

 printStudent(name1, age1, department1, grades1, gpa1); // print John
 printStudent(name2, age2, department2, grades2, gpa2); // print Mary

}
```

There is a better way to model students

- Now, let's use object-oriented approach
- Student information (name, age, printInfo etc.) can be modeled using Student class

```
public static void main(String[] args) {




 String name = "John";
 int age = 21;
 String department = "COMP";
 int[] grades = new int[10];
 double gpa = 2.98;

 printStudent(name, age, department, grades, gpa);
}
```

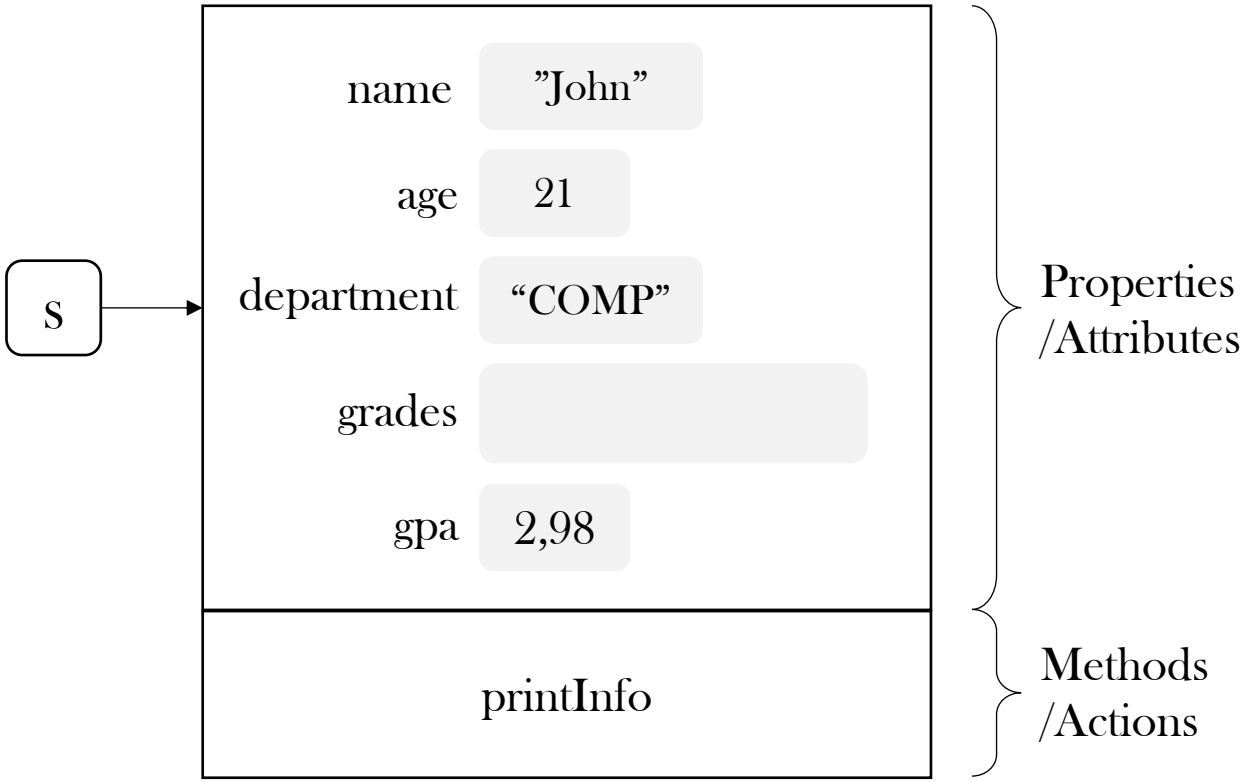
Classical (Procedural) Approach

```
public static void main(String[] args) {

 // let's create an empty student first
 Student s = new Student();

 s.name = "John";
 s.age = 21;
 s.department = "COMP";
 s.gpa = 2.98;
 s.grades = new int[10];

 s.printInfo();
}
```

Object-oriented Approach

There is a better way to model students

- Student class defines properties (name, age, gpa, etc.) and methods (printInfo) of students



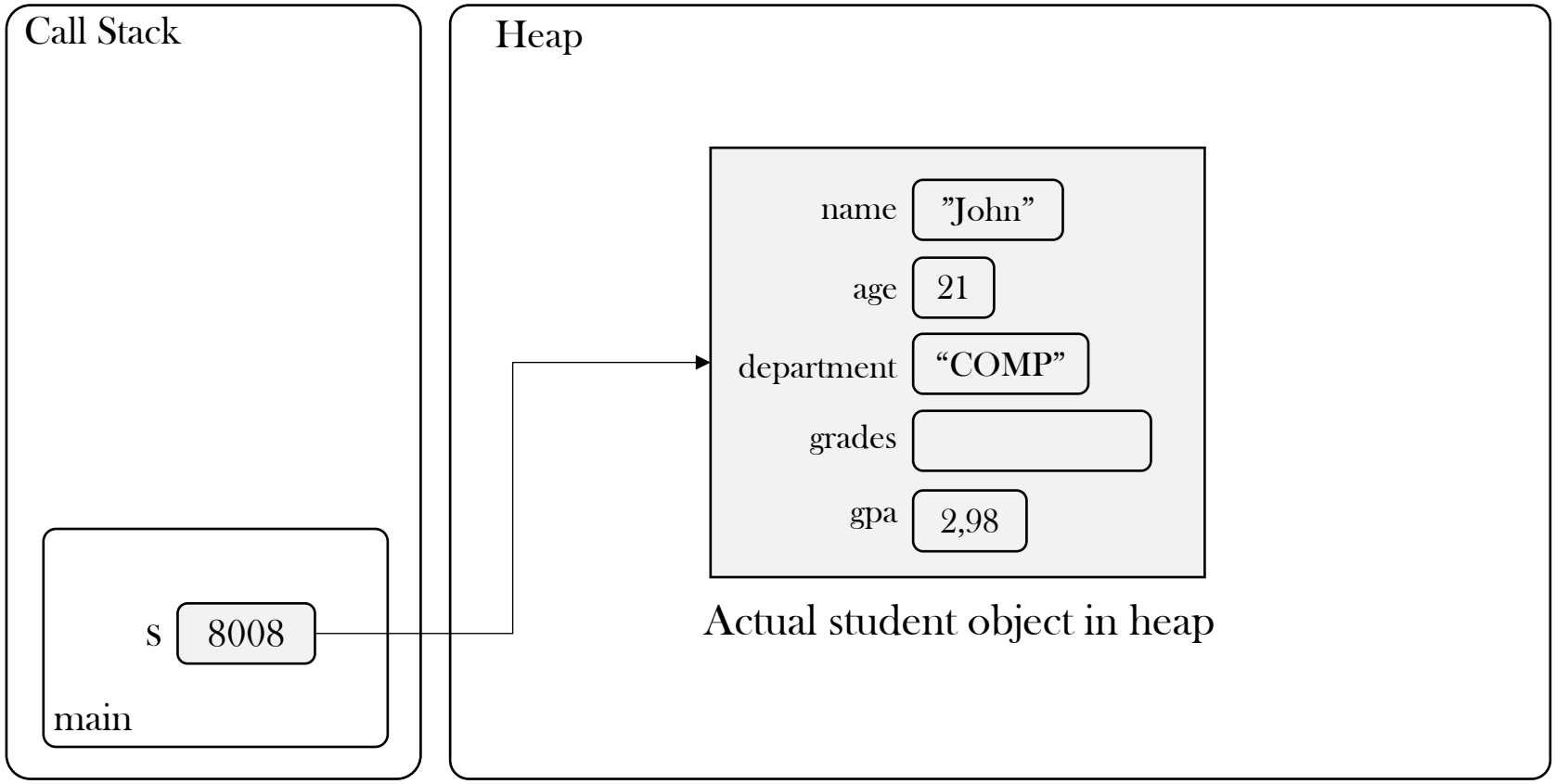

```
public static void main(String[] args) {

 // let's create an empty student first
 Student s = new Student();

 s.name = "John";
 s.age = 21;
 s.department = "COMP";
 s.gpa = 2.98;
 s.grades = new int[10];

 s.printInfo();
}
```

Object-oriented Approach

Student Object and Memory Location

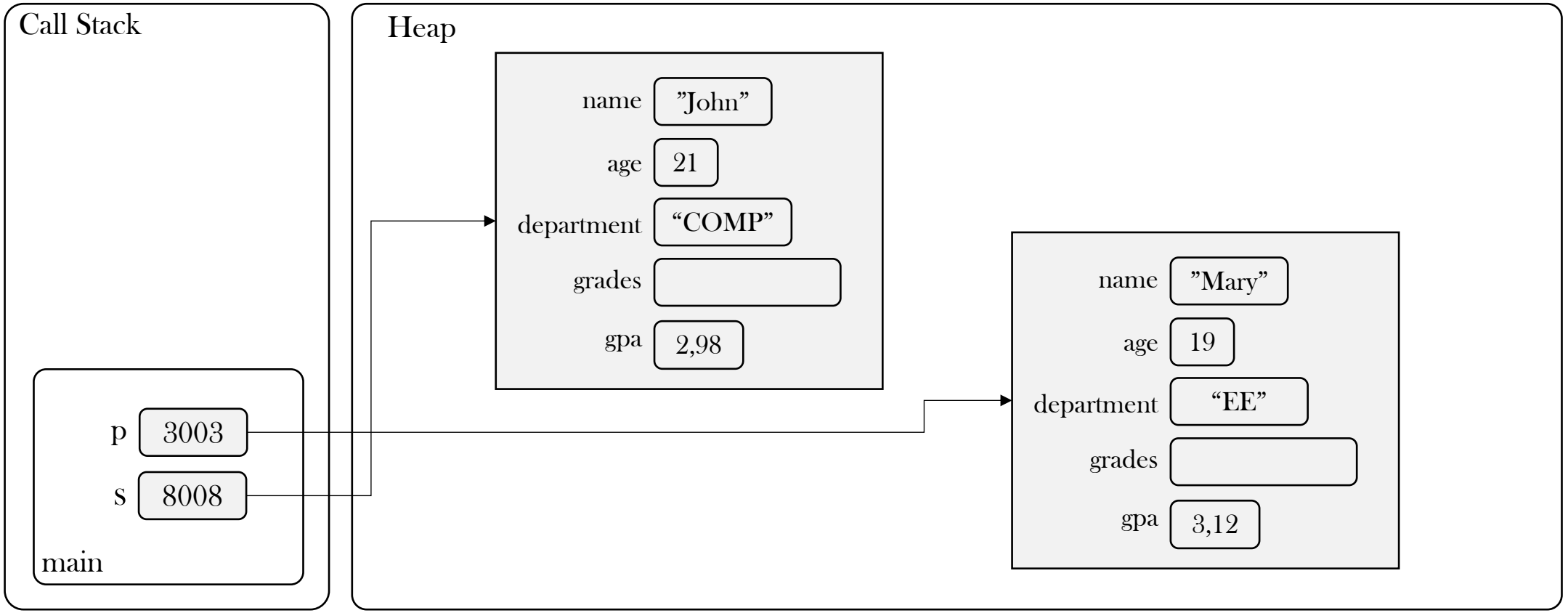
- `s` is an object reference variable and it points to a student object in heap memory

`s` is the object reference variable which points the object in heap



Student Object and Memory Location

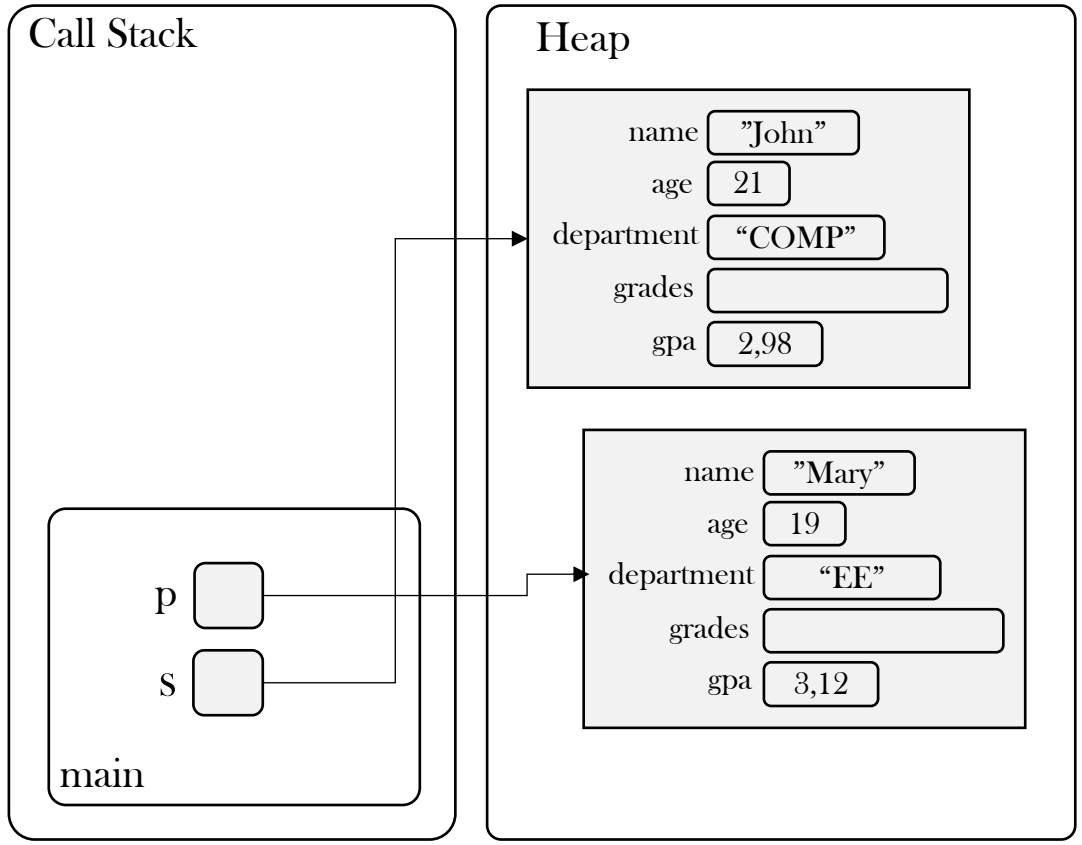
- If we have two students: s and p



Student Objects

- Code to create two students

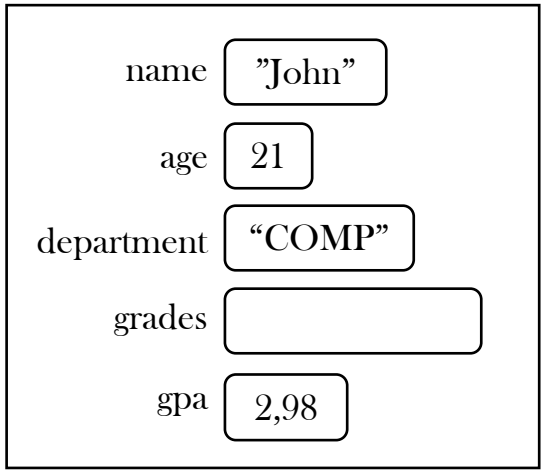
```
public static void main(String[] args) {

 // create a student
 Student s = new Student();
 s.name = "John";
 s.age = 21;
 s.department = "COMP";
 s.gpa = 2.98;
 s.printInfo();

 // create another student
 Student p = new Student();
 p.name = ”Mary";
 p.age = 19;
 p.department = "EE";
 p.gpa = 3.12;
 p.printInfo();
}
```



How to write Student class?

- In order to define properties and methods of students, we have to write the `Student` class

```
// class name
public class Student {

 // class data fields, class instance variables
 public String name;
 public int age;
 public String department;
 public int[] grades;
 public double gpa;

 // constructor: constructor is a special class method
 Student(){
  System.out.println("Creating a new student.");
 }

 // class method(s)
 public void printInfo() { // code omitted }
}
```



Classes and objects

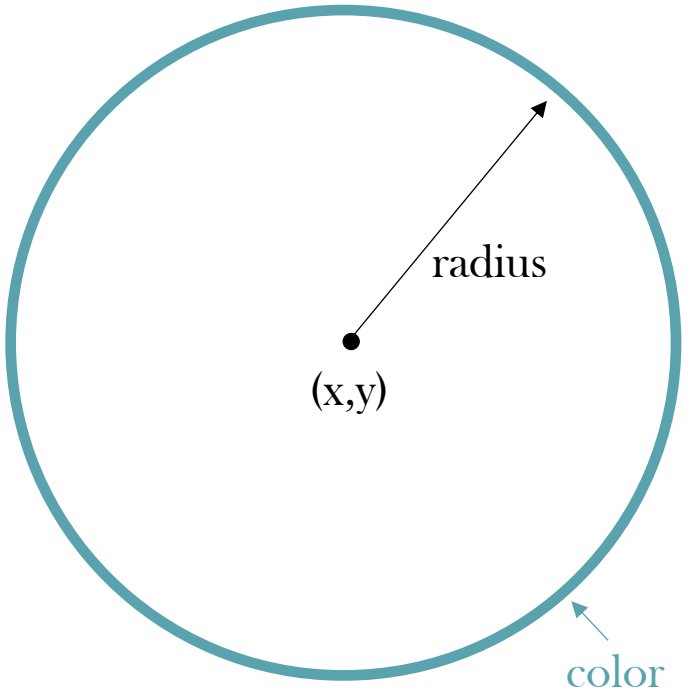
- A class defines the properties and behaviors for objects
  - Property: Data fields (Instance Variables)
  - Behavior: Class methods
- An object is represented by data fields with their current values
  - A rectangle object has the data fields width and height, which characterize a rectangle
- The behavior of an object is defined by methods
  - For example, you may define methods named getArea() and getPerimeter() for circle objects
- A Java class uses
  - Variables to define data fields
  - Methods to define actions

Class Example

Circle

Let's create a circle class

- A circle has the following properties (data fields)
  - Radius
  - (x,y) center coordinates
  - Color
- A circle has the following behaviors/actions (methods)
  - Compute area
  - Compute perimeter
  - Change radius or (x,y) center coordinates
  - Change color
  - Print circle information

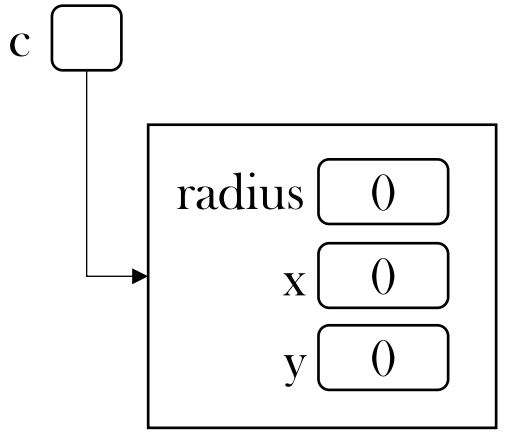


Circle Class

- How to create a circle and use it in main application

```
public class AppCircle {
  public static void main(String[] args) {

    // create a circle using the constructor
    Circle c = new Circle();

    // set properties of the circle using data fields
    c.radius = 3.1; // set radius
    c.x = 2.0; // set x center coordinate
    c.y = 3.5; // set y center coordinate

    // print circle information
    System.out.println("Radius of the circle: " + c.radius);
    System.out.println("Center coordinates of the circle: x=" + c.x + ", y=" + c.y);

    // use getArea class method to print circle area
    System.out.println("Area of the circle: " + c.getArea());
  }
}
```

Creates a circle object using the `new` operator.
`Circle()` is the constructor

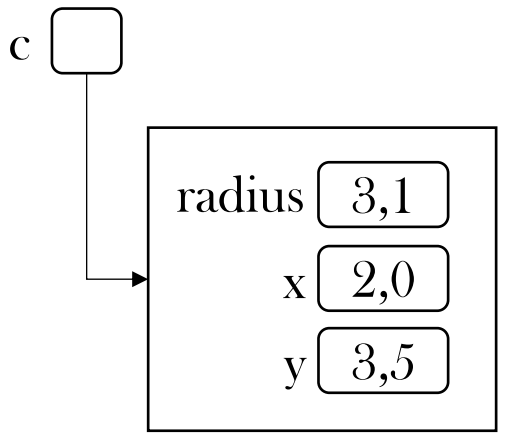


Circle Class

- How to create a circle and use it in main application

```
public class AppCircle {
  public static void main(String[] args) {

    // create a circle
    Circle c = new Circle();

    // set properties of the circle using data fields
    c.radius = 3.1; // set radius
    c.x = 2.0; // set x center coordinate
    c.y = 3.5; // set y center coordinate

    // print circle information
    System.out.println("Radius of the circle: " + c.radius);
    System.out.println("Center coordinates of the circle: x=" + c.x + ", y=" + c.y);

    // use getArea class method to print circle area
    System.out.println("Area of the circle: " + c.getArea());
  }
}
```

You can set circle object's data fields using the dot (.) operator, e.g., `c.radius = 3.1`



Circle Class

- How to create a circle and use it in main application

```
public class AppCircle {
  public static void main(String[] args) {

    // create a circle
    Circle c = new Circle();

    // set properties of the circle using data fields
    c.radius = 3.1; // set radius
    c.x = 2.0; // set x center coordinate
    c.y = 3.5; // set y center coordinate

    // print circle information
    System.out.println("Radius of the circle: " + c.radius);
    System.out.println("Center coordinates of the circle: x=" + c.x + ", y=" + c.y);

    // use getArea class method to print circle area
    System.out.println("Area of the circle: " + c.getArea());
  }
}
```

You can get circle object's data fields using the dot (.) operator, e.g., `c.radius`

Circle Class

- How to create a circle and use it in main application

```
public class AppCircle {
  public static void main(String[] args) {

    // create a circle
    Circle c = new Circle();

    // set properties of the circle using data fields
    c.radius = 3.1; // set radius
    c.x = 2.0; // set x center coordinate
    c.y = 3.5; // set y center coordinate

    // print circle information
    System.out.println("Radius of the circle: " + c.radius);
    System.out.println("Center coordinates of the circle: x=" + c.x + ", y=" + c.y);

    // use getArea class method to print circle area
    System.out.println("Area of the circle: " + c.getArea());
  }
}
```

You can call circle object's methods using the dot (.) operator, e.g., `c.getArea()`

Circle Class – Part 1/2

- We write the circle class in `Circle.java` file:

Data fields (Instance Variables)

Constructors

```
public class Circle {

  // data fields (instance variables)
  public double radius; // circle radius
  public double x; // center x coordinate
  public double y; // center y coordinate

  // constructor 1
  Circle(){
    System.out.println("This constructor creates an default circle");
  }

  // constructor 2
  Circle(double r){
    System.out.println("This constructor creates an circle with radius=r");
    radius = r;
  }

  // code continues
}
```

Circle Class – Part 2/2

- We write the circle class in `Circle.java` file:

```
public class Circle {
  // code continues from here

  // class methods

  // returns the area of circle
  public double getArea() {
    return Math.PI * radius * radius;
  }

  // returns the perimeter of circle
  public double getPerimeter() {
    return 2 * Math.PI * radius;
  }

  // you can set the radius of circle using setRadius method
  public void setRadius(double inputRadius) {
    radius = inputRadius;
  }
} // end of Circle class
```

Class methods

`setRadius` method puts the input argument `inputRadius` into the object's `radius` data field.

Classes are Defined in Their Own File

- Each class is defined in its own file, named as the class itself
  - E.g., Circle class should be written in `Circle.java` file
- Main application can use the Circle class, if they are in the same directory

AppMain.java

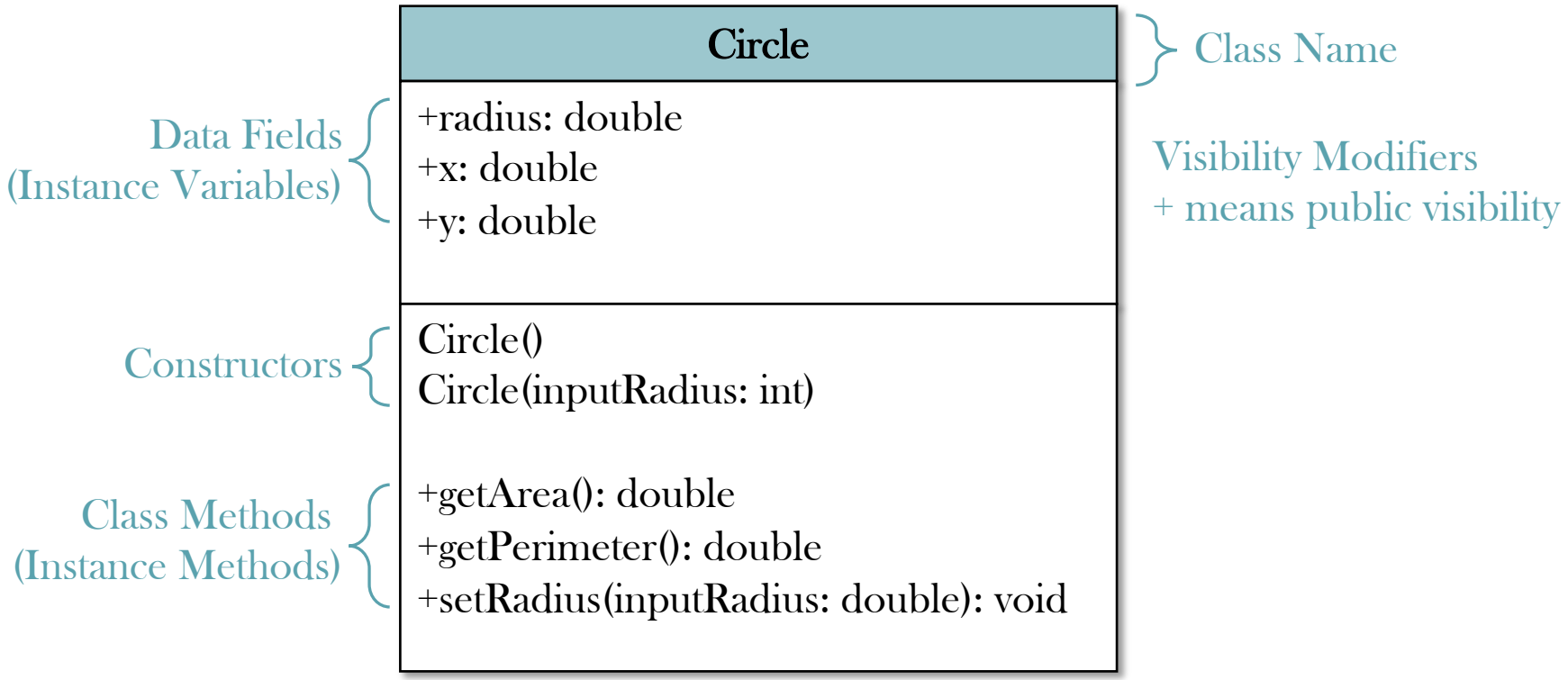
```
public class App {
  public static void main(String[] args) {

    // create a circle
    Circle c = new Circle();

    // set properties of the circle
    c.radius = 3.1; // set radius
    c.x = 2.0; // set x center coordinate
    c.y = 3.5; // set y center coordinate
  }
  . . .
}
```

Circle.java

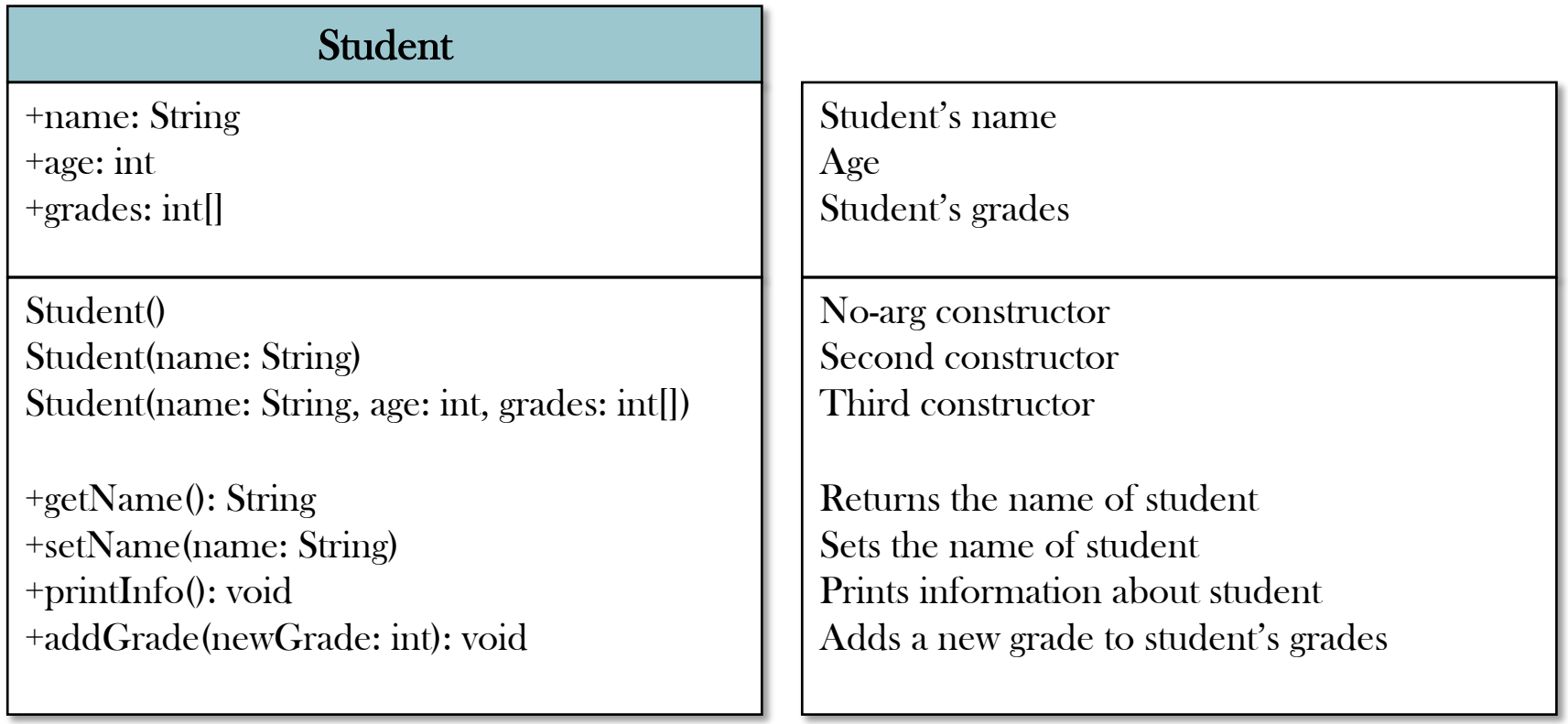
```
public class Circle {
  // data fields (instance variables)
  public double radius; // circle radius
  public double x; // center x coordinate
  public double y; // center y coordinate

  // constructors
  Circle(){
    System.out.println("Creates a circle");
  }
  . . .
}
```

UML Class Diagrams

- Unified Modeling Language (UML) class diagram, provides all information about a class
- Below, UML diagram of Circle class is shown:



UML Class Diagram: Student Class Example



Explanations of data fields and methods

Constructors

Creating objects

Objects

- An object is an instance of a class
- You create an object (instance) of a class
- A class is a template that defines what an object's data fields and methods will be

- How to create objects?
  `Circle circle1 = new Circle();` creates an object `circle1` from the `Circle` class

- Objects are created using constructors: `Circle()` method is a constructor
- Constructors are special class methods

Constructors

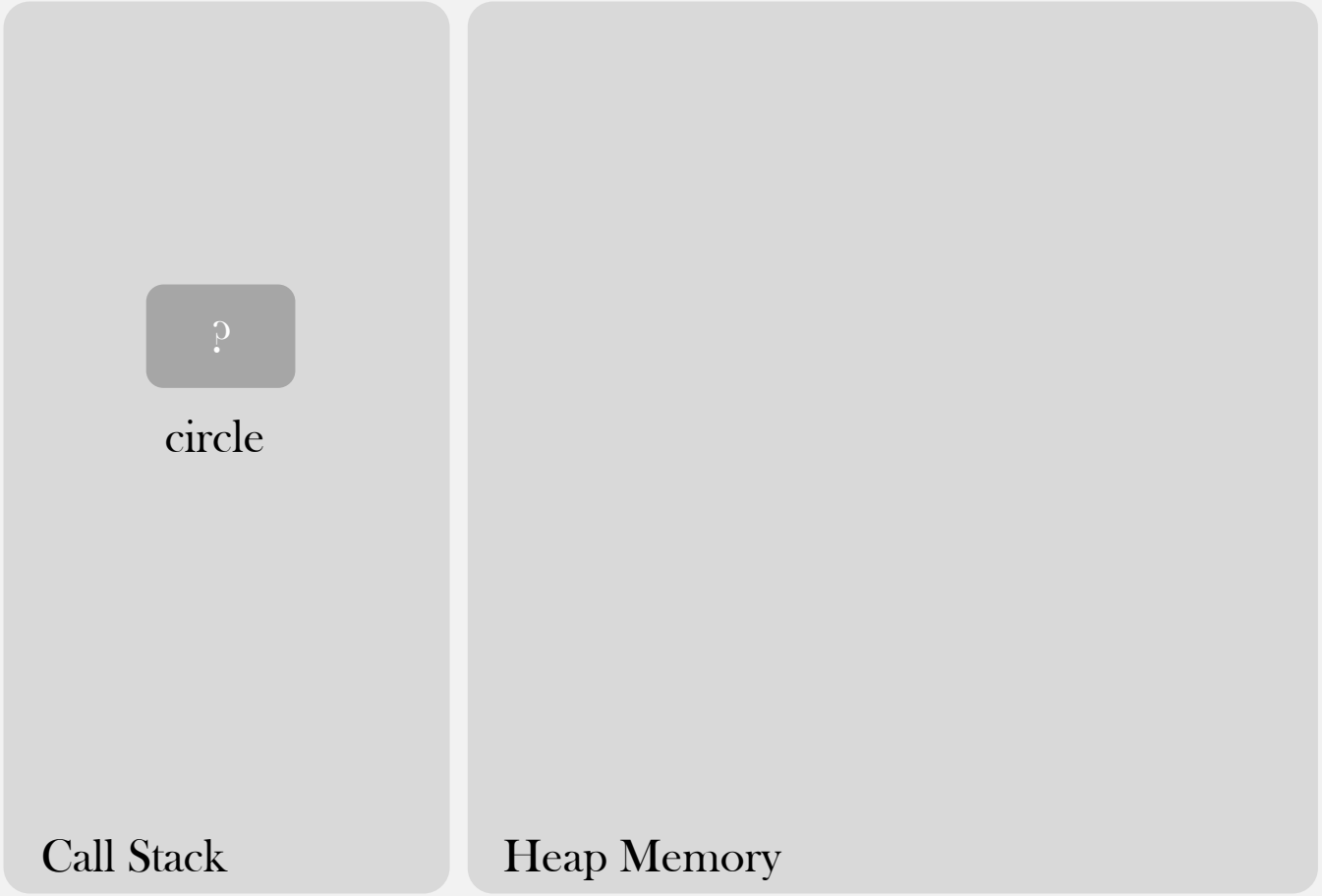
- A class provides methods of a special type, known as constructors, which are invoked to create a new object
- Constructors are designed to perform initializing actions, such as initializing the data fields of objects

- For example:
  `Circle circle1 = new Circle();`
  Here `Circle()` is the constructor: It creates a circle object and initializes it

  `Circle circle1 = new Circle(x,y,radius);`
  Here `Circle(x,y,radius)` is another the constructor:
  It creates a circle object and initializes its center coordinates and radius

Steps of Creating Objects

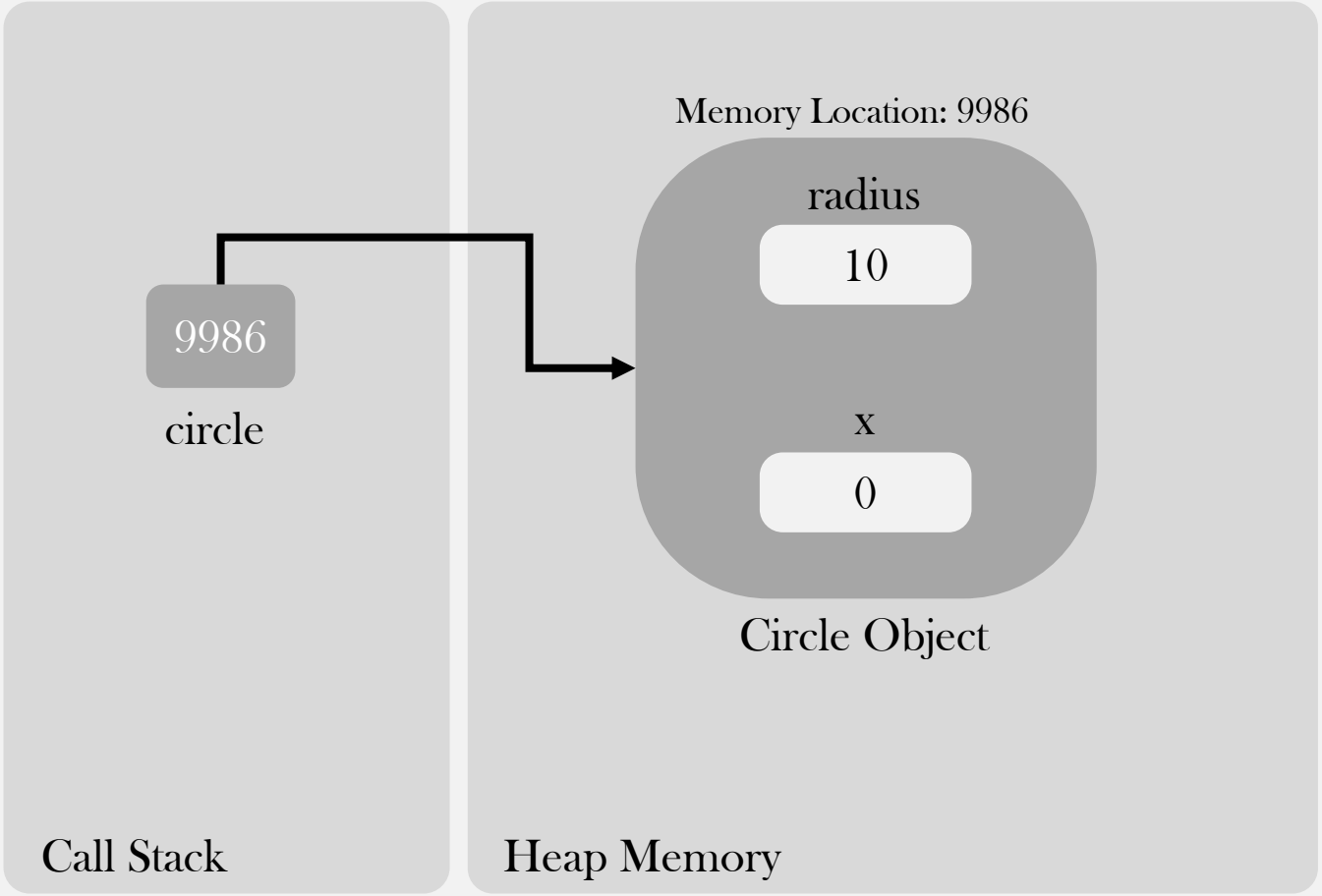
1. Circle circle; // Uninitialized
2. circle = new Circle(10);
3. circle.setX(12);



STEP 1: DECLARING AN OBJECT

Steps of Creating Objects

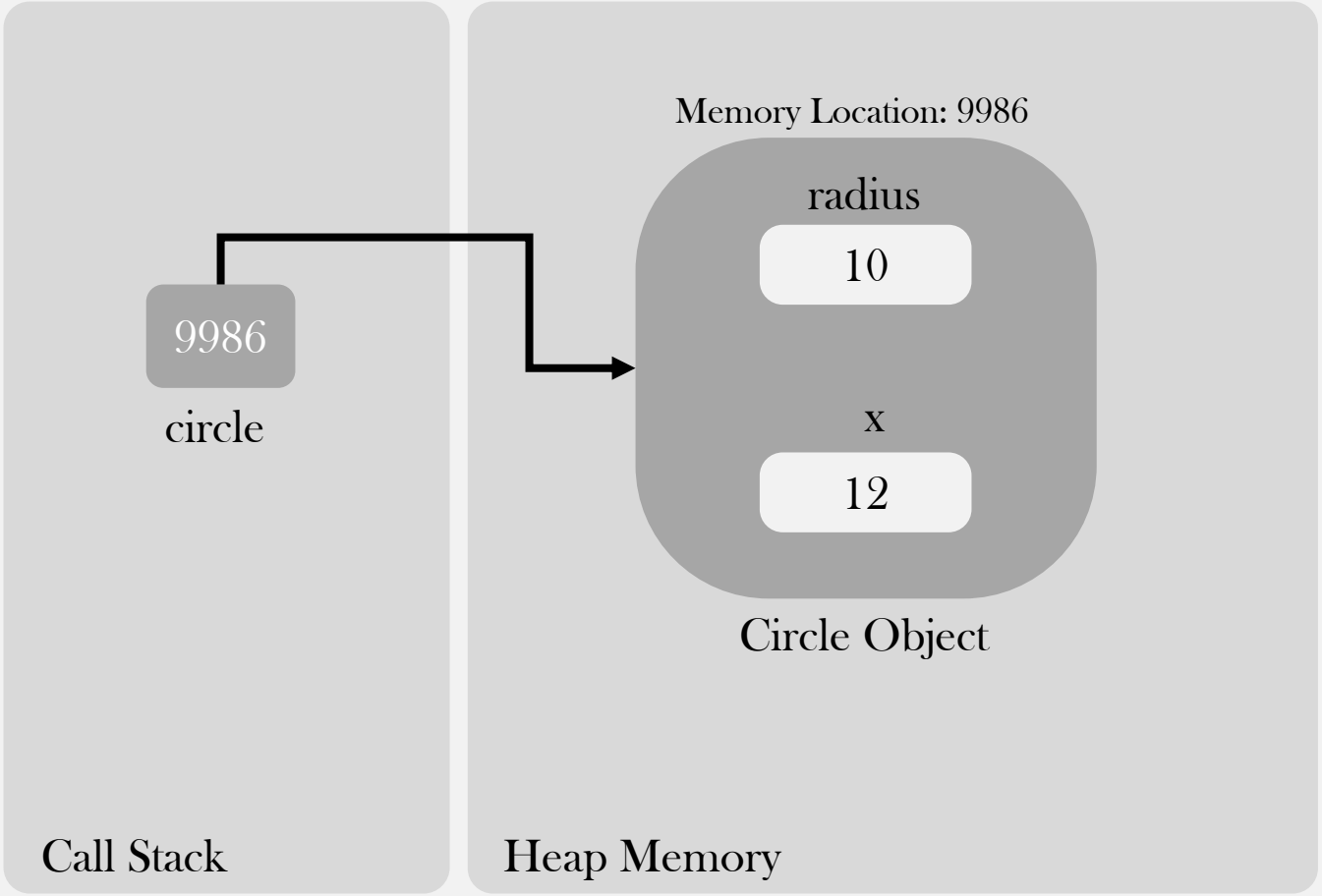
1. Circle circle;
2. circle = new Circle(10);
3. circle.setX(12);



STEP 2: CREATING AND INITIALIZING AN OBJECT

Steps of Creating Objects

1. Circle circle;
2. circle = new Circle(10);
3. circle.setX(12);

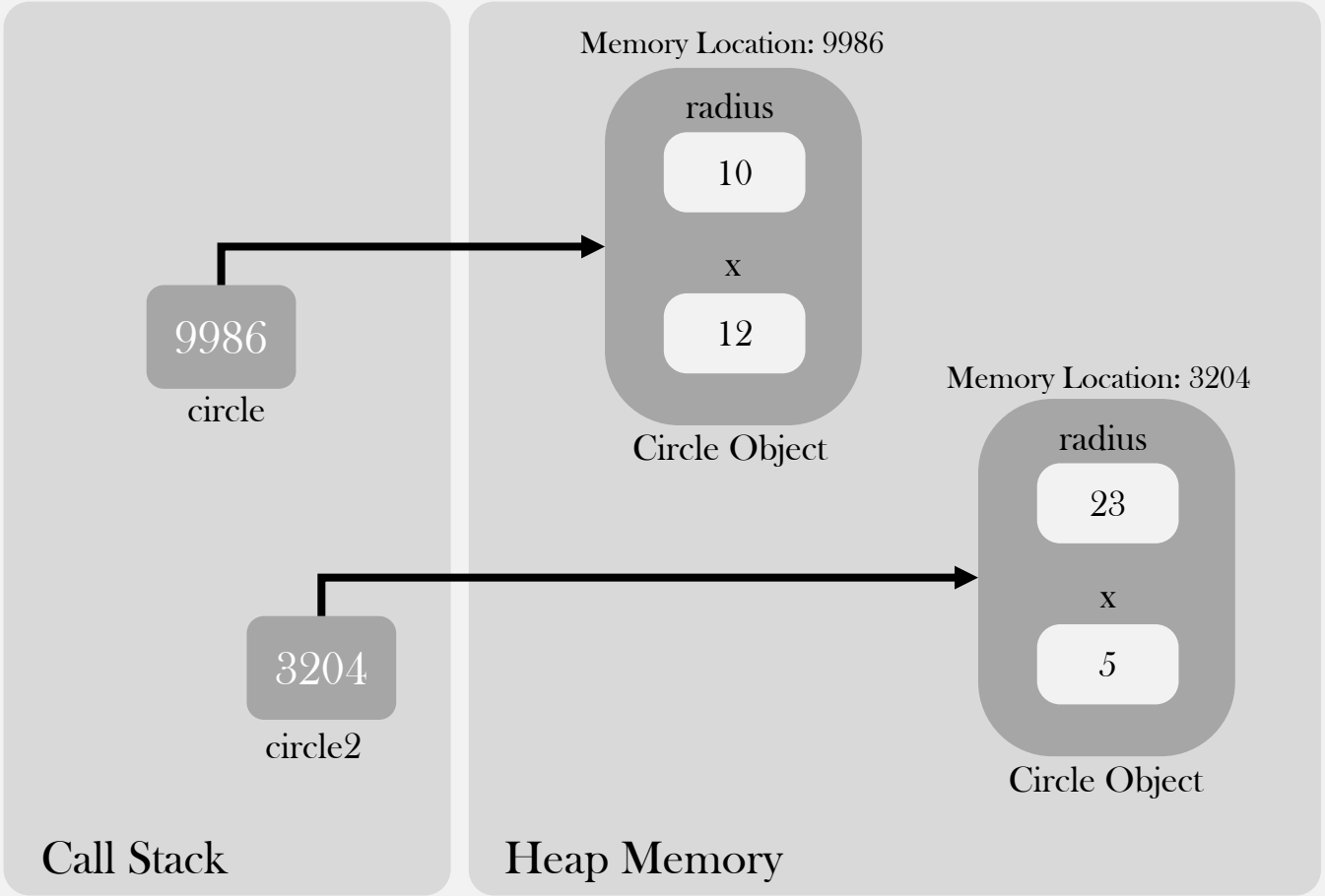


STEP 3: SETTING OBJECT'S VARIABLES BY CALLING ITS METHODS

Steps of Creating Objects

1. Circle circle;
2. circle = new Circle(10);
3. circle.setX(12);

4. Circle circle2;
5. circle2 = new Circle(23,5);



YOU CAN CREATE MANY OBJECTS

Constructors

- Constructors play the role of creating and initializing objects
- Constructor must have the same name as the class itself
- Constructors do not have a return type - not even void

```
public class Circle {

  public double radius;
  public double x, y;

  // Constructor
  Circle(){
    System.out.println("Creates a default circle");
    radius = 10;
    x = 1;
    y = 2;
  }

}
```

```
public class Student {

  public String name;
  public int age;
  public int[] grades;

  // Constructor
  Student(){
    System.out.println("Creates a new student.");
    name = "John";
    age = 19;
    grades = new int[10];
  }

}
```

Constructors

- To construct an object, invoke a constructor of the class using the `new` operator

```
public class AppCircle {

  public static void main(String[] args) {

    // create a circle
    Circle c = new Circle();

  }
}
```

```
public class AppStudent {

  public static void main(String[] args) {

    // create a student
    Student s = new Student();

  }
}
```

Constructors

- You can have multiple constructors with the same name but different signatures
  - Constructors can be overloaded

Circle.java

```
Circle() {
  System.out.println("Creates a default circle");
  radius = 10;
  x = 1;
  y = 2;
}

Circle(double r) {
  System.out.println("Creates a circle with radius");
  radius = r;
  x = 4;
  y = 4;
}

Circle(double r, double xcenter, double ycenter) {
  System.out.println("Creates a circle with radius, x, and y");
  radius = r;
  x = xcenter;
  y = ycenter;
}
```

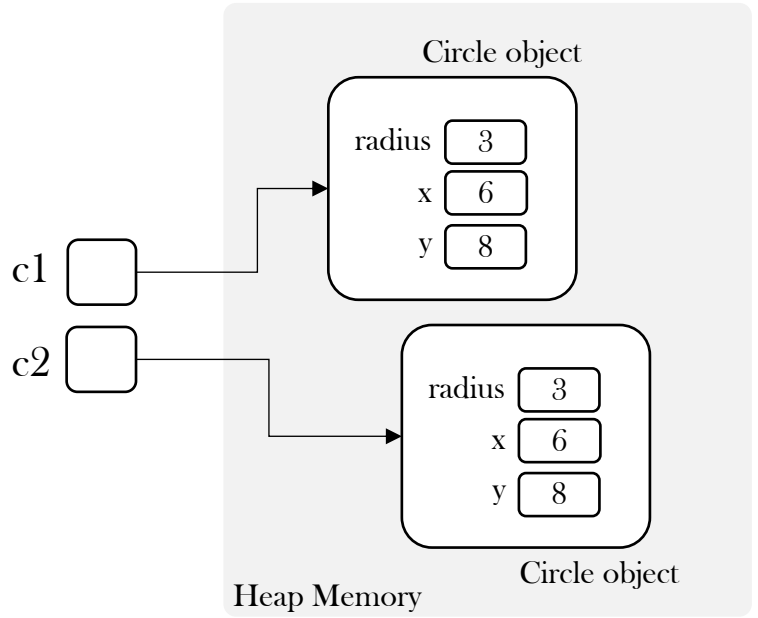
AppCircle.java

```
public class AppCircle {
  public static void main(String[] args) {

    // create a circle with default values
    // calls the first constructor
    Circle c1 = new Circle();

    // create a circle with radius 10
    // calls the second constructor
    Circle c2 = new Circle(10);

    // create a circle with radius 12, x=2, y=4
    // calls the third constructor
    Circle c3 = new Circle(12, 2, 4);
  }
}
```

Constructors

- Overloaded constructor example: Creating a circle with radius=3, x=6, and y=8 using two different ways

```
public class AppCircle {
  public static void main(String[] args) {

    // calls the first constructor
    Circle c1 = new Circle();
    c1.radius = 3;
    c1.x = 6;
    c1.y = 8;

    // calls the third constructor
    Circle c2 = new Circle(3, 6, 8);
  }
}
```



No-arg Constructors and Default Constructors

- A class normally provides a constructor without arguments
  - Such a constructor is referred to as no-argument (no-arg) constructor or default constructor
- A class may be defined without constructors
  - In this case, a public no-arg constructor (default constructor) with an empty body is implicitly defined in the class

```
public class Circle {

  public double radius;
  public double x, y;

  // No-arg constructor (default constructor)
  Circle(){
    System.out.println("Creates a default circle");
    radius = 2;
    x = 2;
    y = 2;
  }
}
```

```
public class Student {

  public String name;
  public int age;
  public int[] grades;

  // No-arg constructor (default constructor)
  Student(){
    System.out.println("Creates a new student.");
    name = "John";
    age = 19;
    grades = new int[10];
  }
}
```

Default Constructor Example

- If you do not write a no-arg constructor, Java automatically creates -an invisible- default for you

Computer.java

```
public class Computer {

  public String name; // e.g., Apple, Razor Blade
  public double cpuSpeed; // e.g., 1.2GHz, 4.2GHz

  // there is no constructor defined by the programmer

}
```


Although `Computer()` constructor is not written, Java automatically creates it with an empty method body. This constructor is invisible.

AppComputer.java

```
public class AppComputer {
  public static void main(String[] args) {

    // create a default computer
    Computer myComputer = new Computer();

    // print information
    System.out.println("Name: " + myComputer.name);
    System.out.println("Speed: " + myComputer.cpuSpeed);

  }
}
```

Program output

```
Computer name: null
CPU Speed: 0.0
```

Always Write Your No-arg Constructors

- Recommendation: Even if the method body of the no-arg constructor is empty, always write your no-arg constructor yourself

```
public class Circle {

 public double radius;
 public double x, y;

 // No-arg constructor (default constructor)
 Circle(){
 }

}
```

this keyword

this

- The keyword `this` refers to the object itself
- You can use `this` to reference the object's instance members

`this.gpa` refers to Student's gpa data field. You can read `this.gpa` as "Student's gpa variable"

```
public class Student {

 public String name;
 public double gpa;

 public void setGPA(int gpa) {
  this.gpa = gpa;
 }
}
```

`gpa` refers to method's input parameter gpa

this

- You can not say `gpa = gpa` in the setGPA method

```
public class Student {

 public String name;
 public double gpa;

 public void setGPA(int gpa) {
  gpa = gpa; // this is wrong: causes ambiguity
 }
}
```

this to Refer Instance Variables

- `this` keyword usually used in constructors to refer to instance variables

```
public class Student {

 public String name;
 public int age;
 public String department;
 public int[] grades;
 public double gpa;

 public Student(String name, int age, String department, int[] grades, double gpa) {
  this.name = name;
  this.age = age;
  this.department = department;
  this.grades = grades;
  this.gpa = gpa;
 }
}
```

this to Call Another Constructor

- `this` keyword can be used to invoke another constructor of the same class

```
public class Circle {

  private double radius;
  private double x,y;

  Circle(double inputR){
    radius = inputR;
    x = 1;
    y = 1;
  }

  Circle(double inputR, double xcenter, double ycenter){
    radius = inputR;
    x = xcenter;
    y = ycenter;
  }
}
```

```
public class Circle {

  private double radius;
  private double x,y;

  Circle(double inputR){
    this(inputR, 1, 1); // call the second constructor
  }

  Circle(double inputR, double xcenter, double ycenter){
    radius = inputR;
    x = xcenter;
    y = ycenter;
  }
}
```

this to Call Another Constructor

- If a class has multiple constructors, it is better to implement them using `this`
- In general, a constructor with fewer arguments can invoke a constructor with more arguments
- `this` statement should appear first in the constructor before any other executable statements

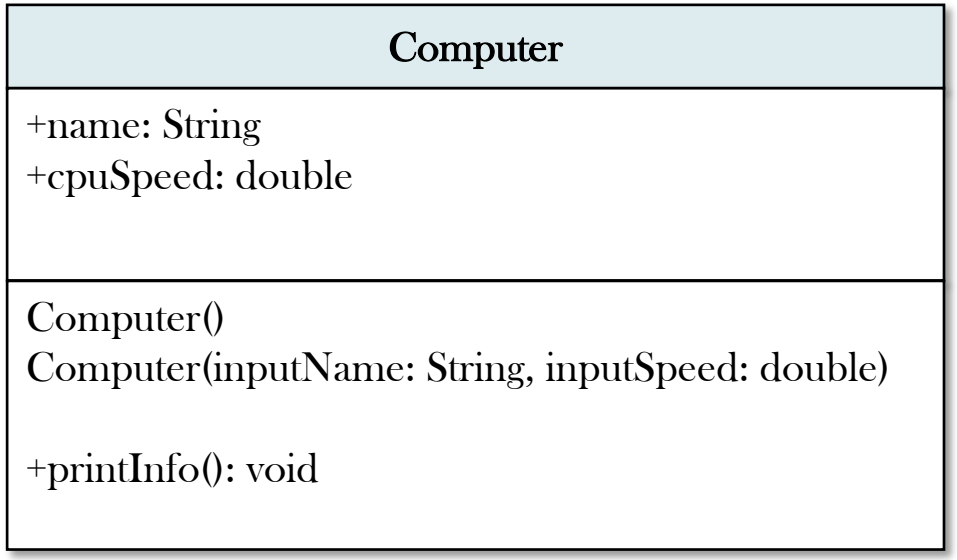
```
Circle(){
 this(10,2,2); // calls the third constructor
}
Circle(double inputR){
 this(inputR, 1, 1); // calls the third constructor
}
Circle(double inputRadius, double xcenter, double ycenter){
 radius = inputRadius;
 x = xcenter;
 y = ycenter;
}
```

Objects

Accessing Data Fields and Methods

- An object's data and methods can be accessed through the dot (.) operator via the object's reference variable
  - In OOP terminology, an object's member refers to its data fields and methods
- In the example below, `c` is an object reference variable

UML Class Diagram of the Computer class

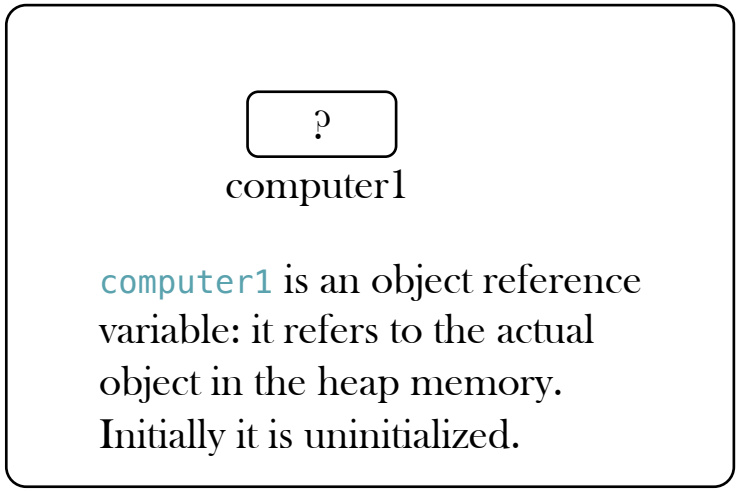



```
// create a computer object
Computer c = new Computer("Apple",3.2);

// change speed
c.cpuSpeed = 1.6;

// print computer name only
System.out.println("Name: " + c.name);

// print all computer information
c.printInfo();
```

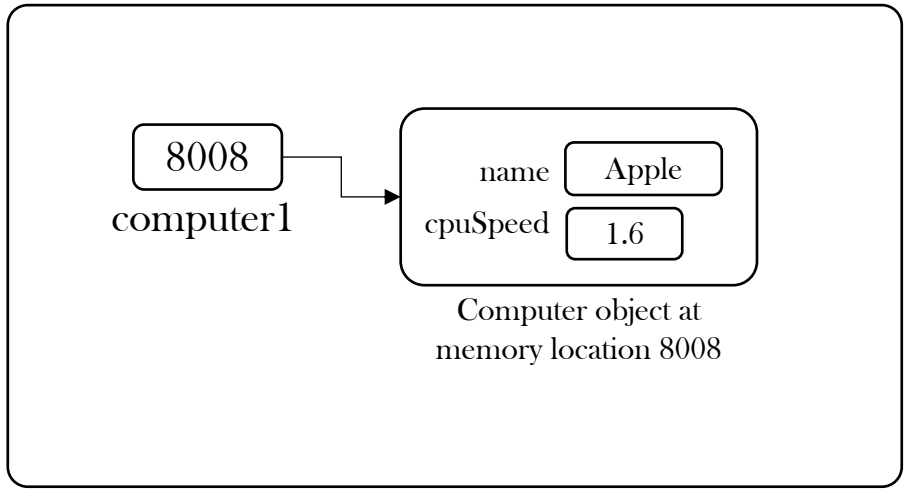
Object Reference Variables

- Objects are accessed via the object's reference variables, which contain references to the objects
  - Object reference variable are declared using the following syntax: `Computer computer1;`
  - Declared objects are not initialized: `computer1` does not contain any value
  - You should use constructors to initialize and create objects, or assign null

```
Computer computer1;
```




```
Computer computer1;
computer1 = new Computer("Apple",1.6);
```



Object Reference Variables

- Statement below declares an object reference variable and then creates an object and assigns its reference to myCircle:

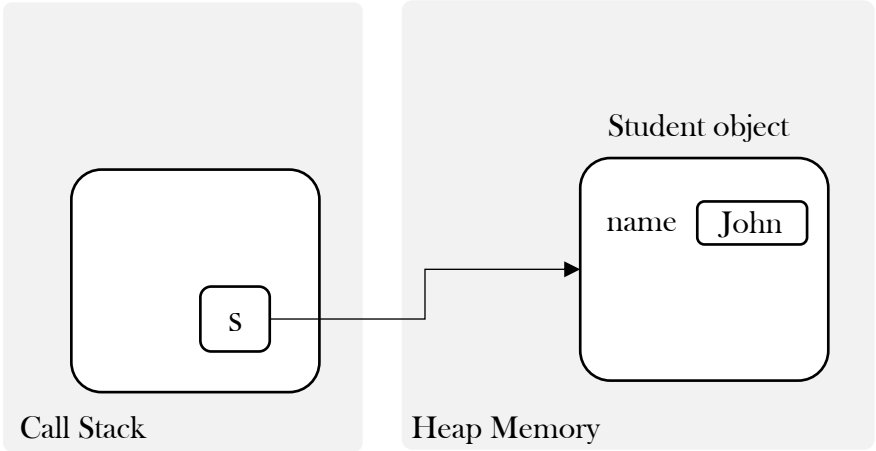
```
Circle myCircle; // declare the object reference variable
myCircle = new Circle(12,0,1); // create the object
```

- You can write a single statement that combines the declaration of an object reference variable, the creation of an object, and the assigning of an object reference to the variable:

```
Circle myCircle = new Circle();
```

Reference Variables

- An object reference variable that appears to hold an object actually contains a reference to that object
  - Object reference variable and an object are different
  - For simplicity, we say that `myCircle` is a Circle object rather than use the long description that `myCircle` is a variable that contains a reference to a Circle object




```
Student s = new Student("John");
```

Which explanation is correct?

1. s is a Student object
2. s is an object reference variable that contains reference to a Student object

Instance variables and methods

- The data field radius is referred to as an instance variable (or data field), because it is dependent on a specific instance
- For the same reason, the method getArea is referred to as an instance method, because you can invoke it only on a specific instance

Data Fields (Instance Variables)

- The data fields of objects can be primitive types (int, double etc.) or reference types (array and object)
- If a data field of a reference type does not reference any object, it holds null literal
- Default values of data fields
  - null for a reference type
  - 0 for a numeric type
  - false for a boolean type

```
Student s = new Student();

System.out.println("Name       : " + s.name);
System.out.println("Age        : " + s.age);
System.out.println("Department : " + s.department);
System.out.println("GPA        : " + s.gpa);
System.out.println("Grades     : " + Arrays.toString(s.grades));
```

Program output

```
Name       : null
Age        : 0
Department : null
GPA        : 0.0
Grades     : null
```

Common Error: Null Pointer Exception

- NullPointerException is a common runtime error and occurs when you try to access object members (instance variables or methods) on a reference variable with a null value

```
public class App {
 public static void main(String[] args) {

  Student t = null;
  System.out.println(t.age); // null pointer exception
  t.printInfoBasic();        // null pointer exception
 }
}
```

Statements cause null pointer exception since student t is not created with a constructor

```
public class App {
 public static void main(String[] args) {

  Student t = new Student("John",20); // correction
  System.out.println(t.age);
  t.printInfoBasic();
 }
}
```

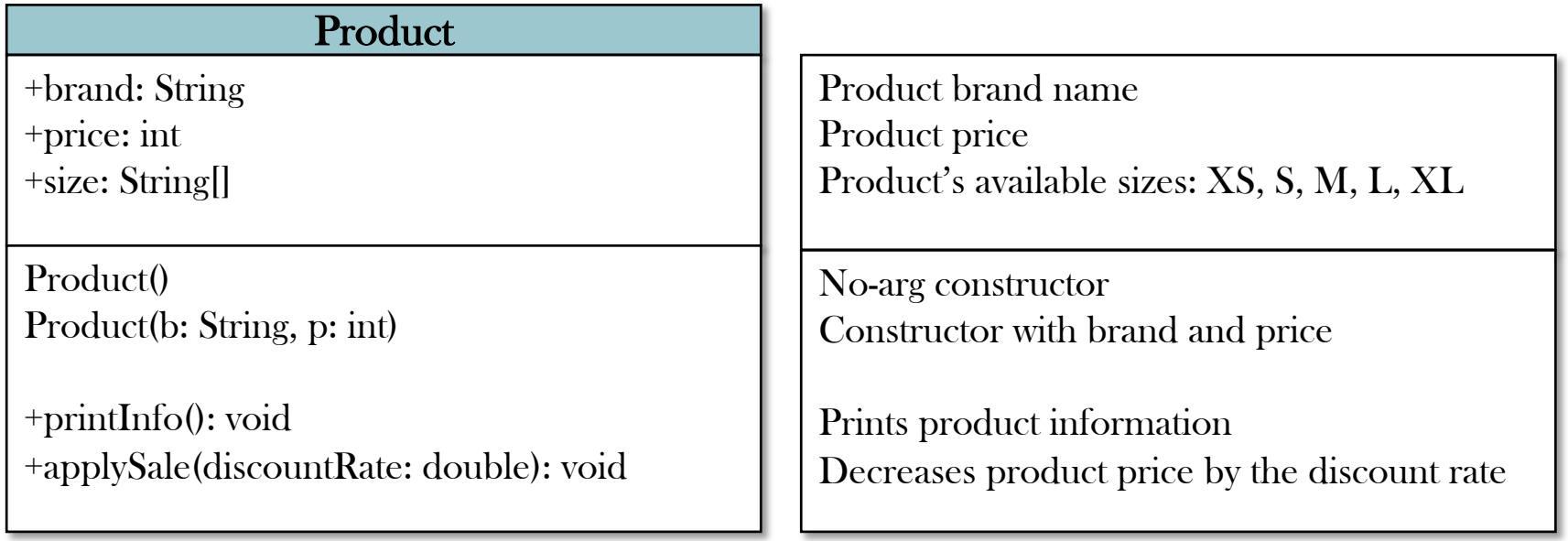
Correction: First, create student using constructor

Class Examples

Product Class

Product Class

- Products have brand name and price
- Prodcut prices decrease when there is a sale, by the discount rate



Use of Product Class in Main

```
public class App {
  public static void main(String[] args) {

    // create a product
    Product p = new Product();
    p.brand = "Adidas";
    p.price = 200.0;
    p.size[0] = "L"; // add a size to a product
    p.size[1] = "XL";

    p.printInfo(); // print product info

    // apply sale to the product: decrease its price
    double discountRate = 0.2;
    p.applySale(discountRate);

    // add a new size to the product
    p.size[2] = "XS";

    p.printInfo();
  }
}
```

Program output

```
Product name   : Adidas
Product price  : 200.0
Product sizes  : [L, XL, null, null, null]


Product name   : Adidas
Product price  : 160.0
Product sizes  : [L, XL, XS, null, null]
```

Product Class

Part 1/2

```
import java.util.Arrays;

public class Product {

  public String brand;   // Adidas, Nike, etc.
  public double price;   // Price

  // a product can have six sizes:
  // extra small (XS), small (S), medium (M), large (L), extra large (XL)
  // size array stores available sizes
  public String[] size = new String[5];   // XS, S, M, L, XL

  /**
   * No-arg constructor
   */
  Product() {}

  /**
   * Constructor: creates a product with brand name and price
   * @param inputBrand Brand name of the product
   * @param inputPrice Price of the product
   */
  Product(String inputBrand, double inputPrice) {
    brand = inputBrand;
    price = inputPrice;
  }
  // code continues
}
```

Part 2/2

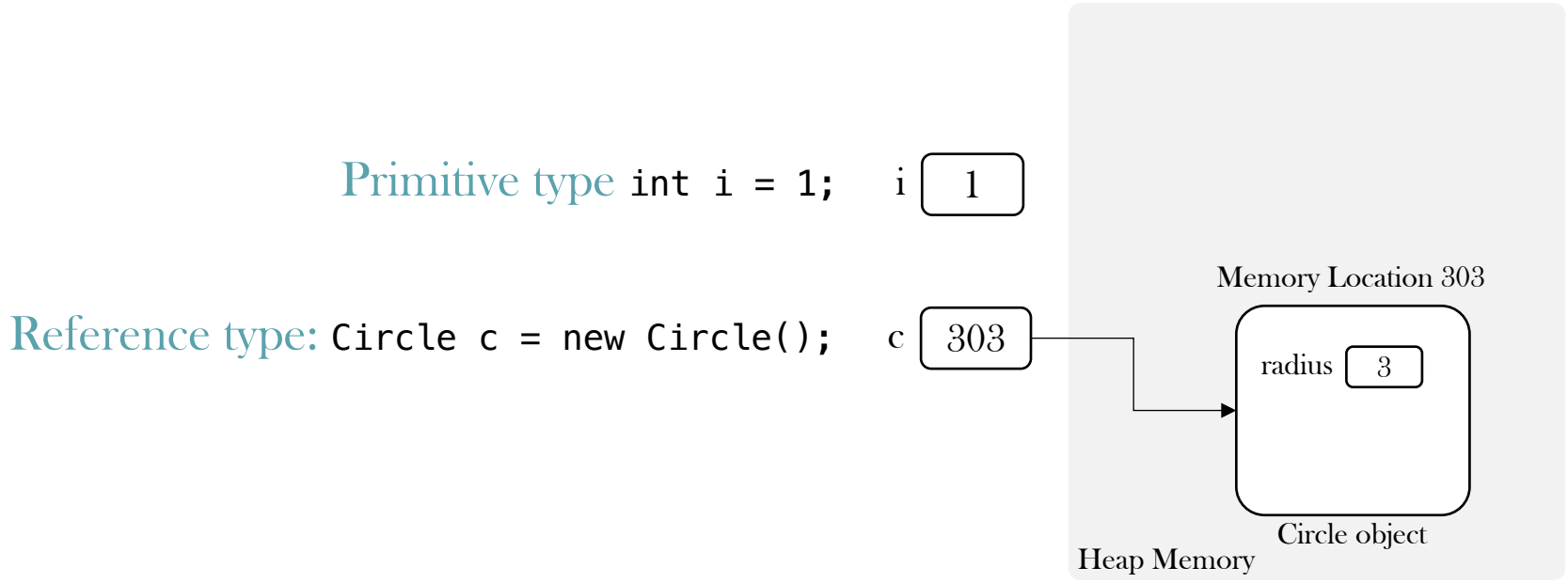
```
  /**
   * Prints product information
   */
  public void printInfo() {
    System.out.println("\nProduct name   : " + brand);
    System.out.println("Product price  : " + price);
    System.out.println("Product sizes  : " + Arrays.toString(size));
  }

  /**
   * Decreases the price of the product by the discount rate
   * Discount rate takes the value between 0 and 1.
   * For example: 0.2 means the discount rate is 20 per cent
   * If the original price is 100, the sales price is 100-(100*0.2)=80
   * @param discountRate Discount rate between 0 and 1.
   */
  public void applySale(double discountRate) {
    price = price * (1-discountRate);
  }
```

Primitive vs Reference Types

Objects

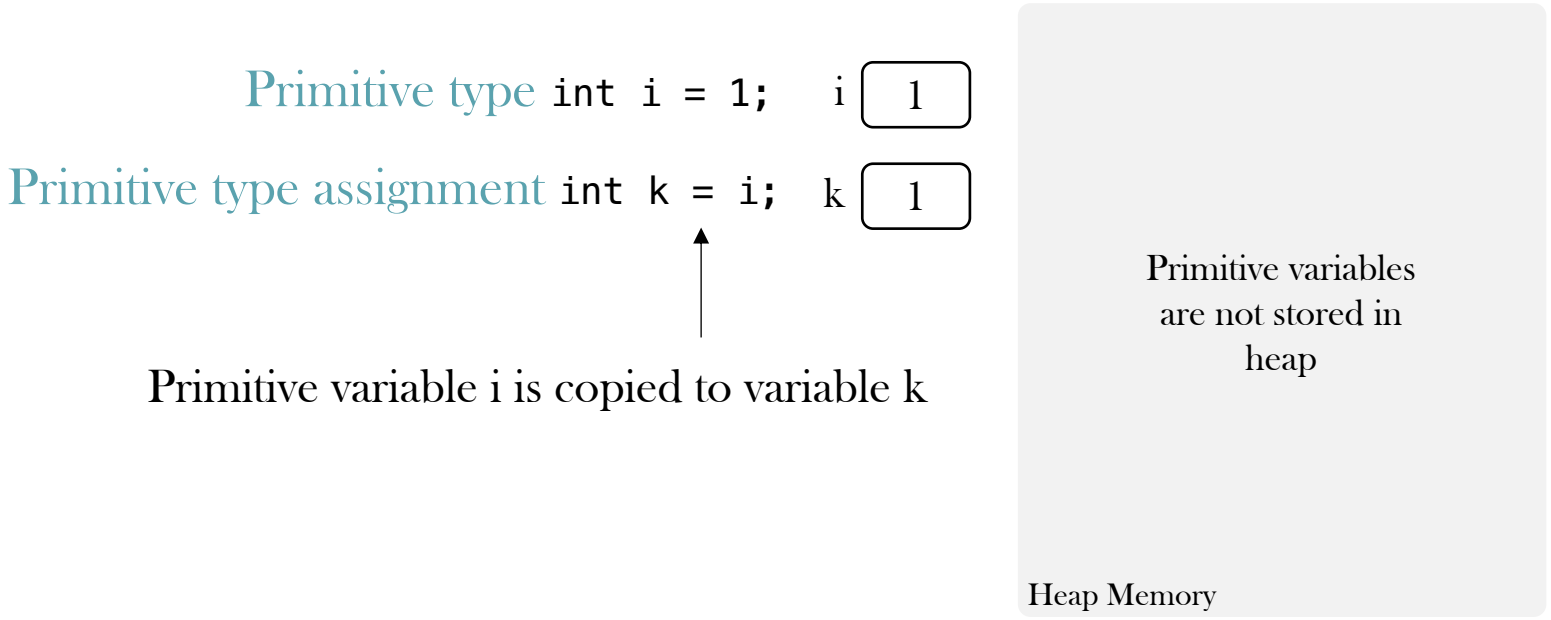
Primitive vs Reference Types

- Every variable represents a memory location that holds a value
- When you declare a variable, you are telling the compiler what type of value the variable can hold
  - For a variable of a primitive type, the value is the primitive type
  - For a variable of a reference type, the value is a reference to where an object is located



Primitive vs Reference Types

- When you assign one variable to another, the other variable is set to the same value
  - For a variable of a primitive type, the real value of one variable is assigned to the other variable
  - For a variable of a reference type, the reference of one variable is assigned to the other variable

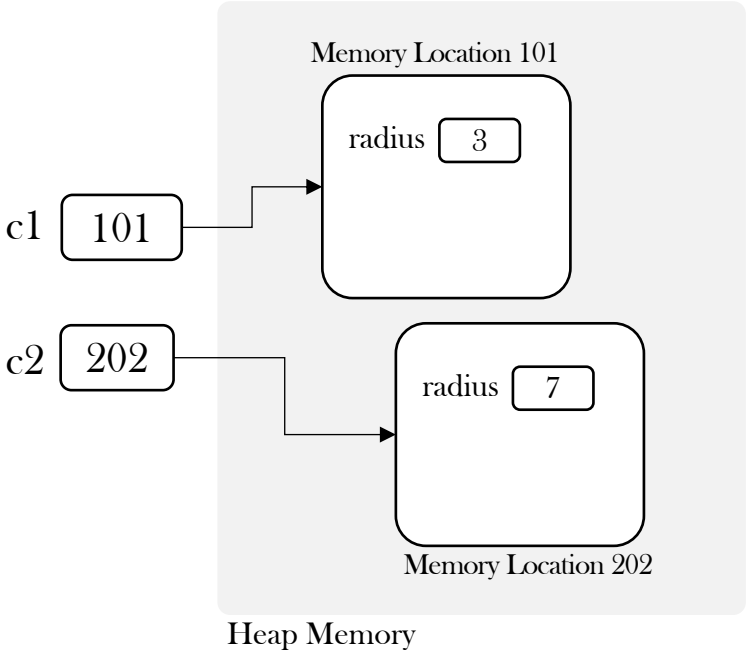


Primitive vs Reference Types

- When you assign one variable to another, the other variable is set to the same value
  - For a variable of a primitive type, the real value of one variable is assigned to the other variable
  - For a variable of a reference type, the reference of one variable is assigned to the other variable

Reference type: `Circle c1 = new Circle(3);`

Reference type: `Circle c2 = new Circle(7);`

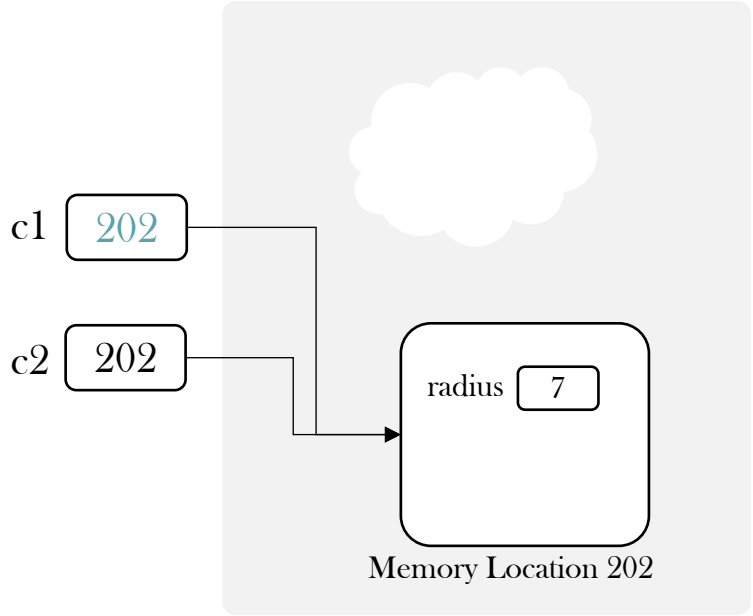


Primitive vs Reference Types

- When you assign one variable to another, the other variable is set to the same value
  - For a variable of a primitive type, the real value of one variable is assigned to the other variable
  - For a variable of a reference type, the reference of one variable is assigned to the other variable

When you execute: `c1 = c2;`

Reference variable c2 is copied to variable c1

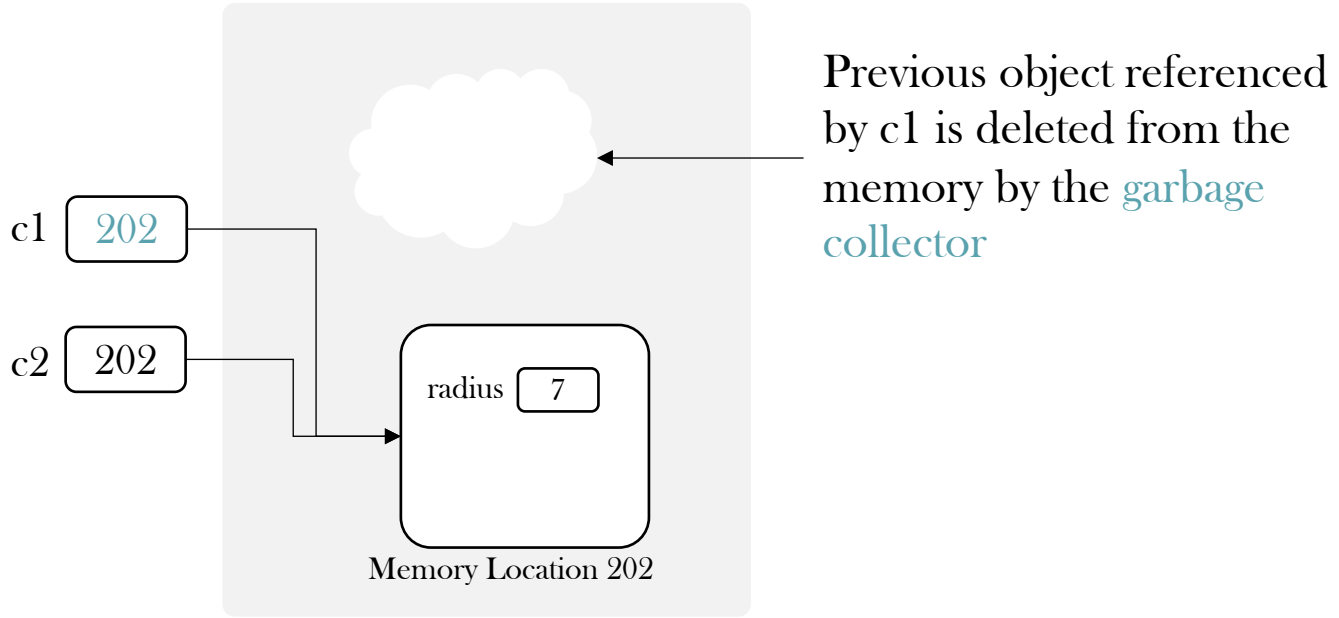


Garbage Collection

- After the assignment statement `c1 = c2`, `c1` points to the same object referenced by `c2`
- The object referenced by `c1` is no longer useful and therefore is now known as garbage
- Garbage occupies memory, so the Java automatically reclaims the space it occupies
- This process is called garbage collection

When you execute: `c1 = c2;`

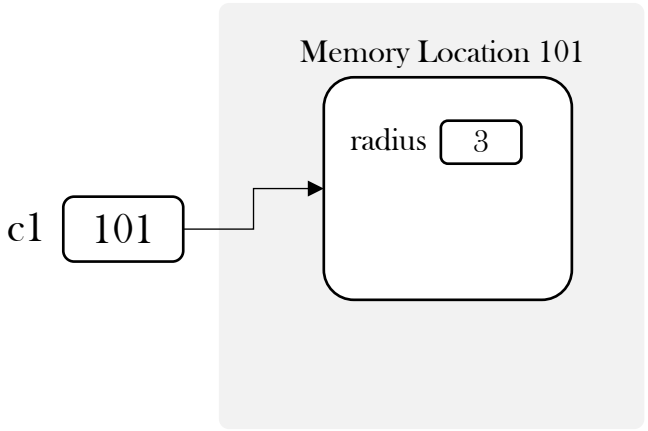
Reference variable c2 is copied to variable c1



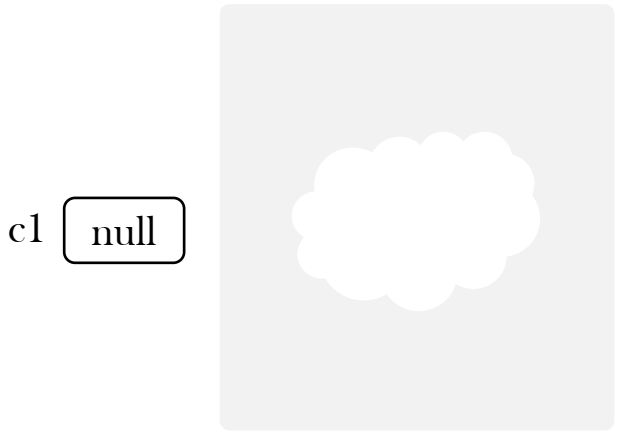
Garbage Collection

- If you know that an object is no longer needed, you can explicitly assign `null` to a reference variable for the object
- The JVM will automatically collect the space if the object is not referenced by any reference variable

```
Circle c1 = new Circle(3);
```




```
c1 = null;
```



Object deleted by the garbage collector

Visibility Modifiers

Public and Private

Visibility modifiers

- Visibility modifiers can be used to specify the visibility of a class and its members
- A visibility modifier specifies how data fields and methods in a class can be accessed from outside the class
- You can use the public visibility modifier for classes, methods, and data fields to denote that they can be accessed from any other classes
- The private modifier makes methods and data fields accessible only from within its own class

We restrict access to the `radius` variable from the main program by making it private

```
public class Circle {

 private double radius;
 public double x,y;

 Circle(double r){
  radius = r;
 }
}
```

```
public static void main(String[] args) {

 Circle c = new Circle();
 c.radius = 5; // this is not allowed: x is private

 c.x = 7; // this is allowed: x is public
}
```

Data field encapsulation

- Making data fields private protects data and makes the class easy to maintain
- To prevent direct modifications of data fields, you should declare the data fields private
- This is known as data field encapsulation

Example:
`pi` variable is only used in Circle class and we do not want it to be visible from the main program. Such internal variables can be defined as private

We prevent direct modification to `pi` variable from outside of the class (data encapsulation)

```
public class Circle {

 public double radius;
 public double x,y;
 private double pi = 3.14;

 Circle(){ }

 public double getArea() {
  return pi * radius * radius;
 }
}
```

Data field encapsulation

- However, a client (main program) often needs to retrieve and modify a data field
  - To make a private data field accessible, provide a getter method (accessors) to return its value
  - To enable a data field to be updated, provide a setter method (mutators) to set a new value

Circle.java

```
public class Circle {

  private double radius;
  public double x,y;

  // setter method
  public void setRadius(double inputRadius) {
    if (radius > 0)
      radius = inputRadius;
    else
      System.out.println("Radius should be a positive!");
  }

  // getter method
  public double getRadius() {
    return radius;
  }
}
```

App.java

```
public static void main(String[] args) {

  Circle c = new Circle();
  c.setRadius(6); // set the circle radius

  // print circle radius
  System.out.println("Radius is " + c.getRadius());
}
```

Data field encapsulation

- Data field encapsulation protects data
  - Example: Circle radius can not be zero or a negative number. Setter method can check these constraints

```
// setter method
public void setRadius(double inputRadius) {
 if (radius > 0)
  radius = inputRadius;
 else
  System.out.println("Radius should be a positive!");
}
```

toString Method

Getting Object Information Using toString

- Suppose you want to print information about an object
- You can write a class method such as printInfo()

```
public class Circle {
 private double radius;
 private double x,y;

 public void printInfo() {
  System.out.println("Radius: " + radius + ", x= " + x + ", y= " + y);
 }
}
```

- Call printInfo method in main to print information about an object

```
Circle myCircle = new Circle(10, 1, 1); // create a circle
myCircle.printInfo() // print circle information
```

Getting Object Information Using toString

- Java provides a special method, `toString()`, which returns a string that can be used to print information about an object
  - Programmer determines the string returned by the `toString` method
- Example:
  - Let's say that circle information string to be generated is: `"Radius: 10, x: 4, y:8"`
  - Circle class should be written as shown below (left)
  - In main, you can simply call `System.out.println(c)` as shown below (right)

```
public class Circle {
 private double radius;
 private double x,y;

 public String toString() {
  String infoString = "Radius: " + radius + ", x: " + x + ", y: " + y;
  return infoString;
 }
}
```

```
public class App {
 public static void main(String[] args) {

  Circle c = new Circle(8,2,2);
  System.out.println(c);

 }
}
```

toString Example

- Write toString method to display information about students

```
public class Student {

  public String name;
  public int age;
  public String department;
  public int[] grades;
  public double gpa;

  @Override
  public String toString() {
    return "Student [name=" + name +
       ", age=" + age +
       ", department=" + department +
       ", grades=" + Arrays.toString(grades) +
       ", gpa=" + gpa + "]";
  }
}
```

```
public class App {
  public static void main(String[] args) {

    Student p = new Student("John", 19, "COMP", 3.12);
    System.out.println(p);

    }
}
```

Program output

```
Student [name=John, age=19, department=COMP, grades=[0, 0, 0, 0, 0], gpa=3.12]
```

Passing Objects to Methods

Passing Objects to Methods

- Passing an object to a method passes the reference of the object to the method
- Since objects are passed to methods using pass by reference, they can be modified

```
public class App {
 public static void main(String[] args) {

  Circle circle1 = new Circle(8);
  Circle circle2 = new Circle(14);
  printTwoCircles(circle1, circle2);
 }

 /**
  * Prints areas of two circles
  * @param circle1 First circle
  * @param circle2 Second circle
  */
 private static void printTwoCircles(Circle circle1, Circle circle2) {
  System.out.println("Area of the first circle  : " + circle1.getArea());
  System.out.println("Area of the second circle : " + circle2.getArea());
 }
}
```

Passing Objects to Methods

- Since objects are passed to methods using pass by reference, they can be modified

```
public class App {
  public static void main(String[] args) {

    // create a circle, centered at x:4, y:5 with radius 10
    Circle circle = new Circle(10,4,5);
    circle.printInfo();
    moveToOrigin(circle); // move the circle center to (x:0, y:0)
    System.out.println("After method call");
    circle.printInfo();
  }

  /**
   * Moves the input circle center to (x:0,y:0)
   * @param inputCircle Input circle object
   */
  private static void moveToOrigin(Circle inputCircle) {
    inputCircle.setX(0);
    inputCircle.setY(0);
  }
}
```

moveToOrigin method can modify the properties of the input object

Call Stack and Heap Status

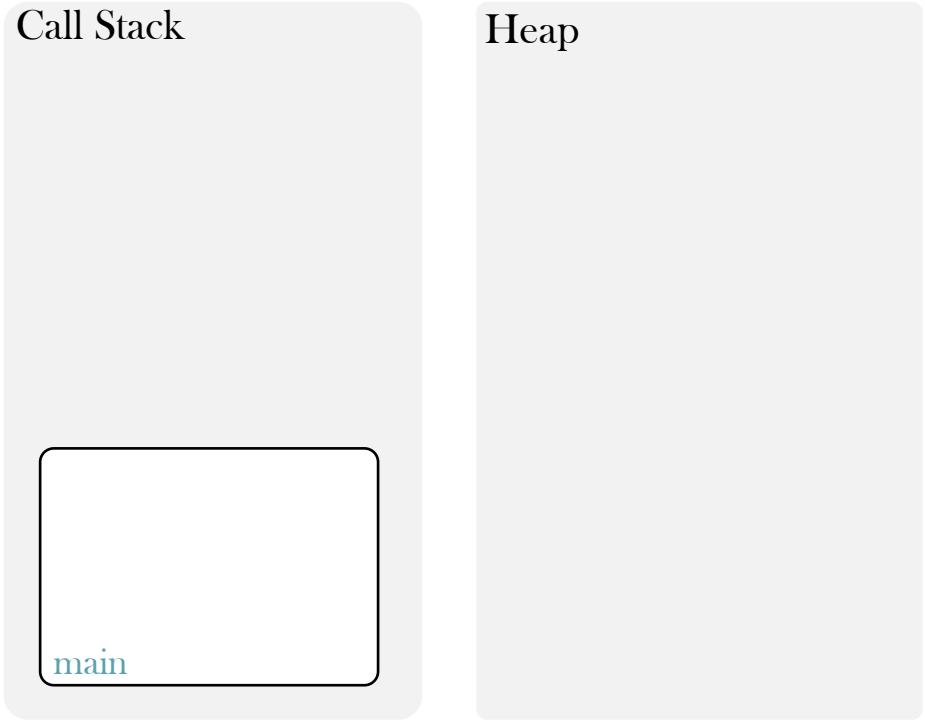
```
1  public class AppCircle {
2    public static void main(String[] args) {
3
4      // create a circle, centered at x:4, y:5 with radius 10
5      Circle circle = new Circle(10,4,5);
6      circle.printInfo();
7      moveToOrigin(circle); // move the circle center to (x:0, y:0)
8      System.out.println("After method call");
9      circle.printInfo();
10   }
11
12   /**
13    * Moves the input circle center to (x:0,y:0)
14    * @param inputCircle Input circle object
15    */
16   private static void moveToOrigin(Circle inputCircle) {
17     inputCircle.setX(0);
18     inputCircle.setY(0);
19   }
20 }
```

Call Stack

Heap

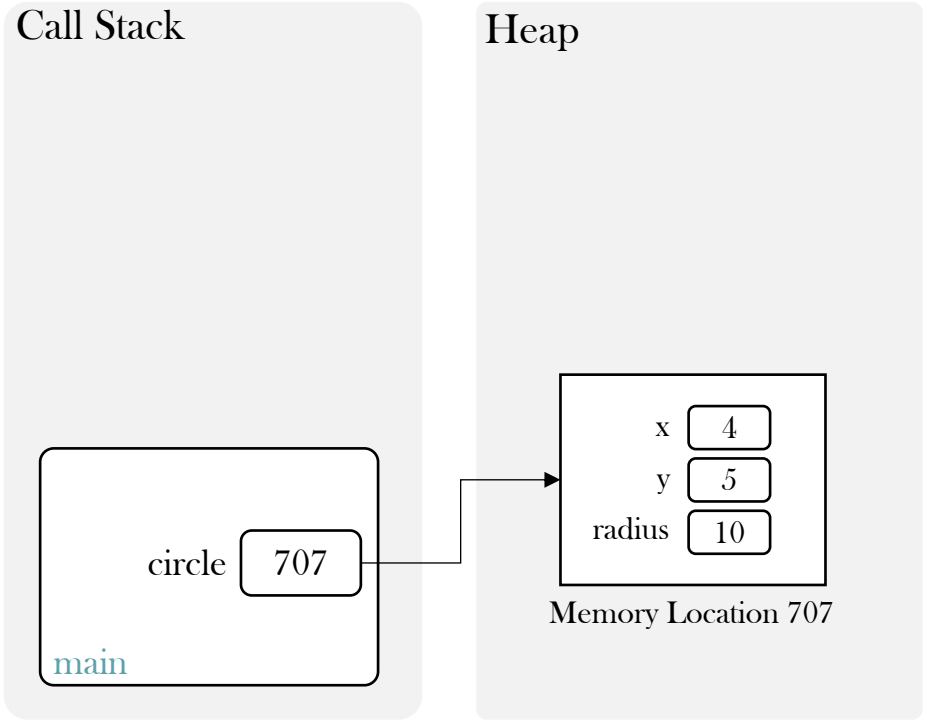
Call Stack and Heap Status

```
1  public class AppCircle {
2    public static void main(String[] args) {
3
4      // create a circle, centered at x:4, y:5 with radius 10
5      Circle circle = new Circle(10,4,5);
6      circle.printInfo();
7      moveToOrigin(circle); // move the circle center to (x:0, y:0)
8      System.out.println("After method call");
9      circle.printInfo();
10   }
11
12   /**
13    * Moves the input circle center to (x:0,y:0)
14    * @param inputCircle Input circle object
15    */
16   private static void moveToOrigin(Circle inputCircle) {
17     inputCircle.setX(0);
18     inputCircle.setY(0);
19   }
20 }
```



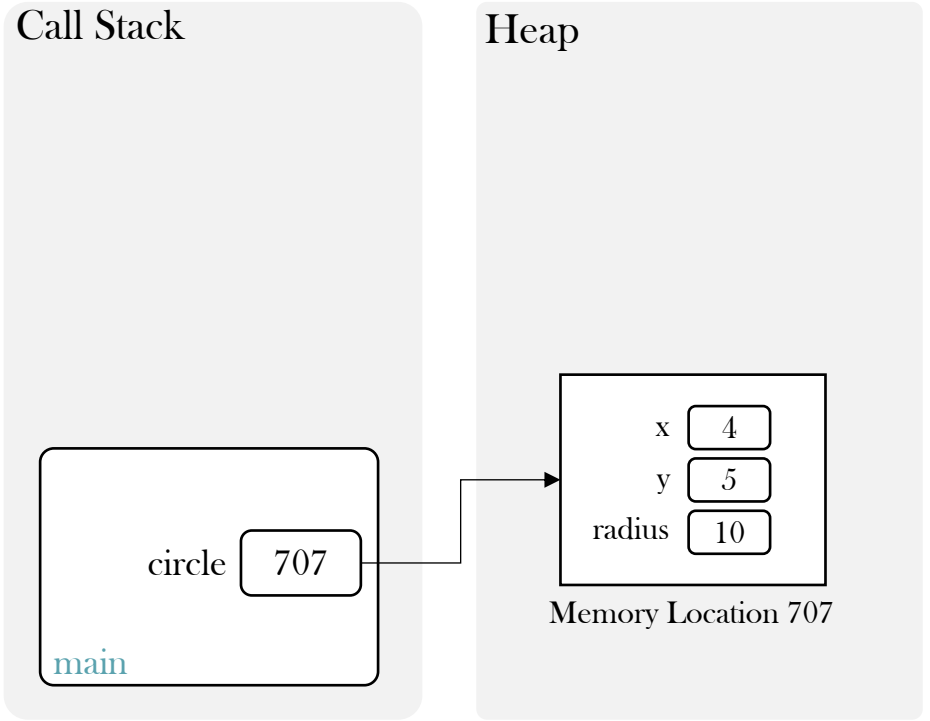
Call Stack and Heap Status

```
1  public class AppCircle {
2    public static void main(String[] args) {
3
4      // create a circle, centered at x:4, y:5 with radius 10
5      Circle circle = new Circle(10,4,5);
6      circle.printInfo();
7      moveToOrigin(circle); // move the circle center to (x:0, y:0)
8      System.out.println("After method call");
9      circle.printInfo();
10   }
11
12   /**
13    * Moves the input circle center to (x:0,y:0)
14    * @param inputCircle Input circle object
15    */
16   private static void moveToOrigin(Circle inputCircle) {
17     inputCircle.setX(0);
18     inputCircle.setY(0);
19   }
20 }
```



Call Stack and Heap Status

```
1  public class AppCircle {
2    public static void main(String[] args) {
3
4      // create a circle, centered at x:4, y:5 with radius 10
5      Circle circle = new Circle(10,4,5);
6      circle.printInfo();
7      moveToOrigin(circle); // move the circle center to (x:0, y:0)
8      System.out.println("After method call");
9      circle.printInfo();
10   }
11
12   /**
13    * Moves the input circle center to (x:0,y:0)
14    * @param inputCircle Input circle object
15    */
16   private static void moveToOrigin(Circle inputCircle) {
17     inputCircle.setX(0);
18     inputCircle.setY(0);
19   }
20 }
```

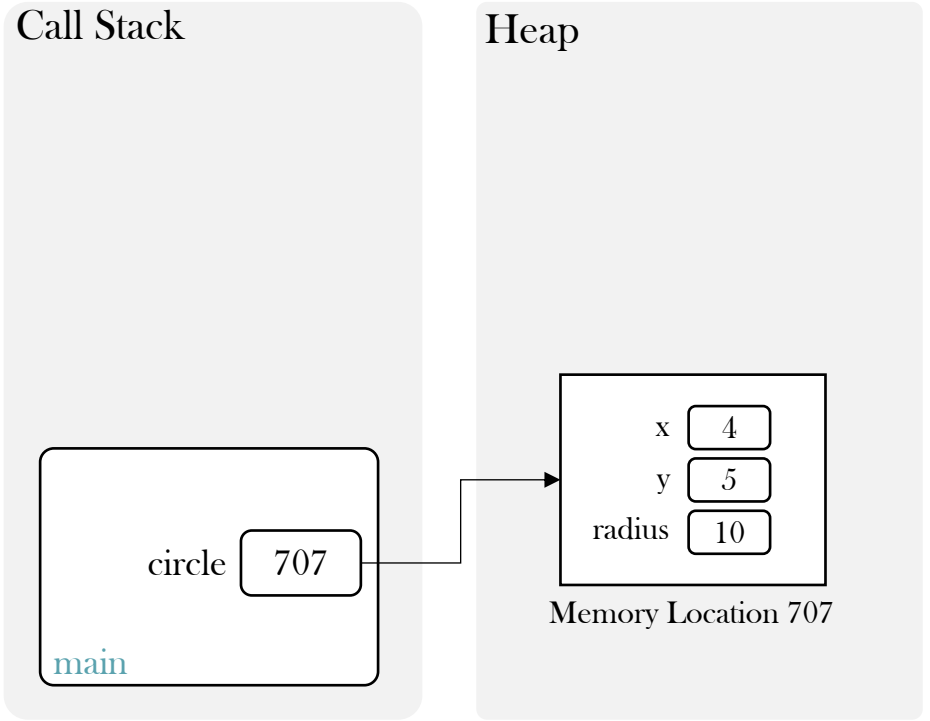


Program output

```
Radius: 10.0, x= 4.0, y= 5.0
```

Call Stack and Heap Status

```
1  public class AppCircle {
2    public static void main(String[] args) {
3
4      // create a circle, centered at x:4, y:5 with radius 10
5      Circle circle = new Circle(10,4,5);
6      circle.printInfo();
7      moveToOrigin(circle); // move the circle center to (x:0, y:0)
8      System.out.println("After method call");
9      circle.printInfo();
10   }
11
12   /**
13    * Moves the input circle center to (x:0,y:0)
14    * @param inputCircle Input circle object
15    */
16   private static void moveToOrigin(Circle inputCircle) {
17     inputCircle.setX(0);
18     inputCircle.setY(0);
19   }
20 }
```

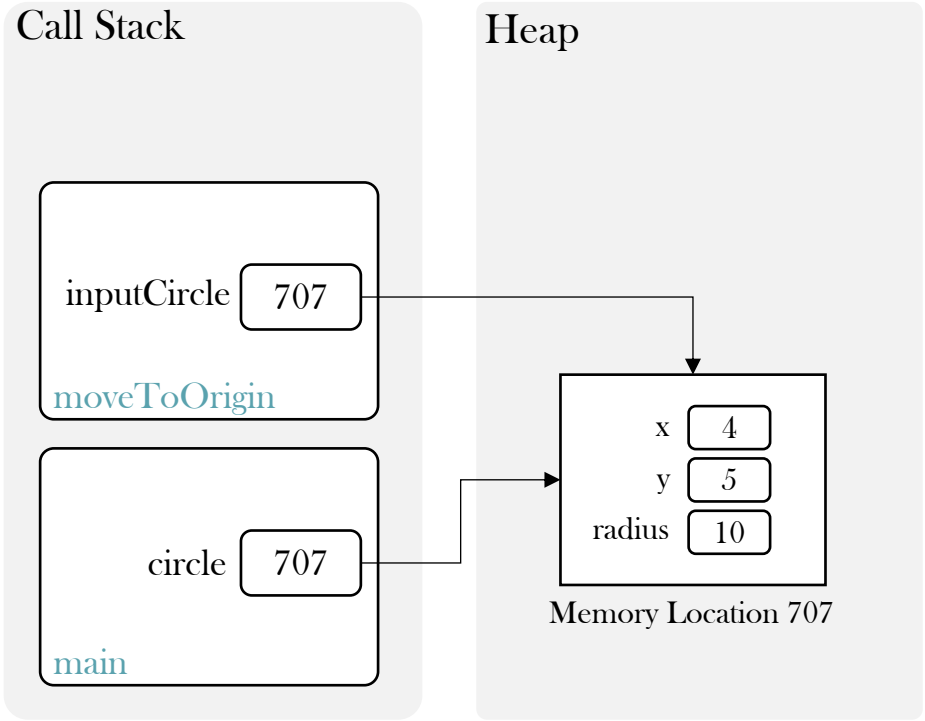


Program output

```
Radius: 10.0, x= 4.0, y= 5.0
```

Call Stack and Heap Status

```
1  public class AppCircle {
2    public static void main(String[] args) {
3
4      // create a circle, centered at x:4, y:5 with radius 10
5      Circle circle = new Circle(10,4,5);
6      circle.printInfo();
7      moveToOrigin(circle); // move the circle center to (x:0, y:0)
8      System.out.println("After method call");
9      circle.printInfo();
10   }
11
12   /**
13    * Moves the input circle center to (x:0,y:0)
14    * @param inputCircle Input circle object
15    */
16   private static void moveToOrigin(Circle inputCircle) {
17     inputCircle.setX(0);
18     inputCircle.setY(0);
19   }
20 }
```

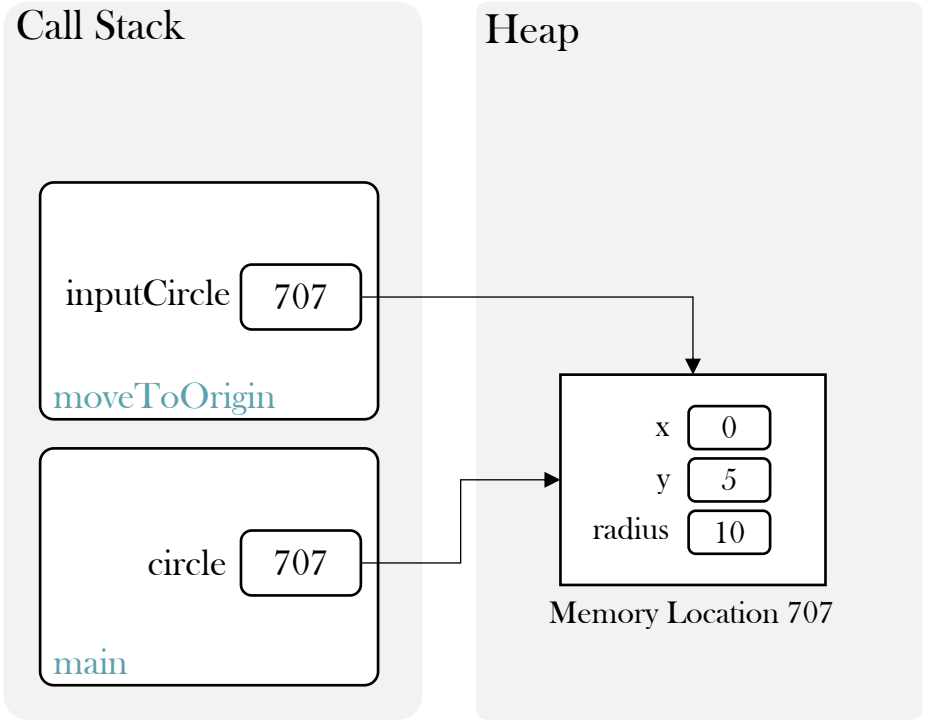


Program output

```
Radius: 10.0, x= 4.0, y= 5.0
```

Call Stack and Heap Status

```
1  public class AppCircle {
2    public static void main(String[] args) {
3
4      // create a circle, centered at x:4, y:5 with radius 10
5      Circle circle = new Circle(10,4,5);
6      circle.printInfo();
7      moveToOrigin(circle); // move the circle center to (x:0, y:0)
8      System.out.println("After method call");
9      circle.printInfo();
10   }
11
12   /**
13    * Moves the input circle center to (x:0,y:0)
14    * @param inputCircle Input circle object
15    */
16   private static void moveToOrigin(Circle inputCircle) {
17     inputCircle.setX(0);
18     inputCircle.setY(0);
19   }
20 }
```



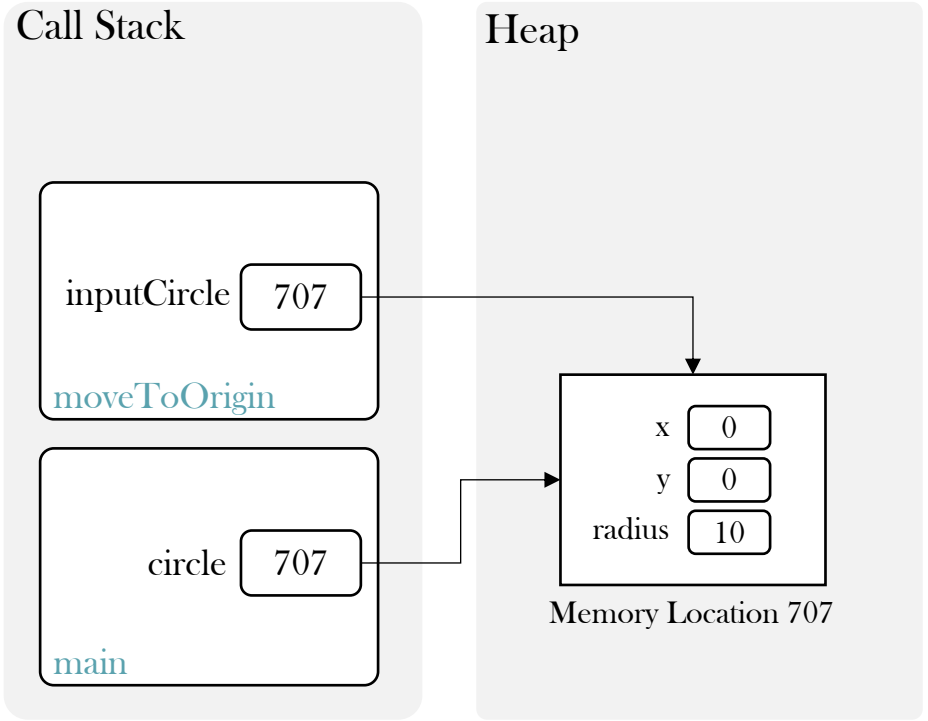
Program output

```
Radius: 10.0, x= 4.0, y= 5.0
```

You can modify data fields of the original object inside the method since inputCircle refers to the circle object in main

Call Stack and Heap Status

```
1  public class AppCircle {
2    public static void main(String[] args) {
3
4      // create a circle, centered at x:4, y:5 with radius 10
5      Circle circle = new Circle(10,4,5);
6      circle.printInfo();
7      moveToOrigin(circle); // move the circle center to (x:0, y:0)
8      System.out.println("After method call");
9      circle.printInfo();
10   }
11
12   /**
13    * Moves the input circle center to (x:0,y:0)
14    * @param inputCircle Input circle object
15    */
16   private static void moveToOrigin(Circle inputCircle) {
17     inputCircle.setX(0);
18     inputCircle.setY(0);
19   }
20 }
```

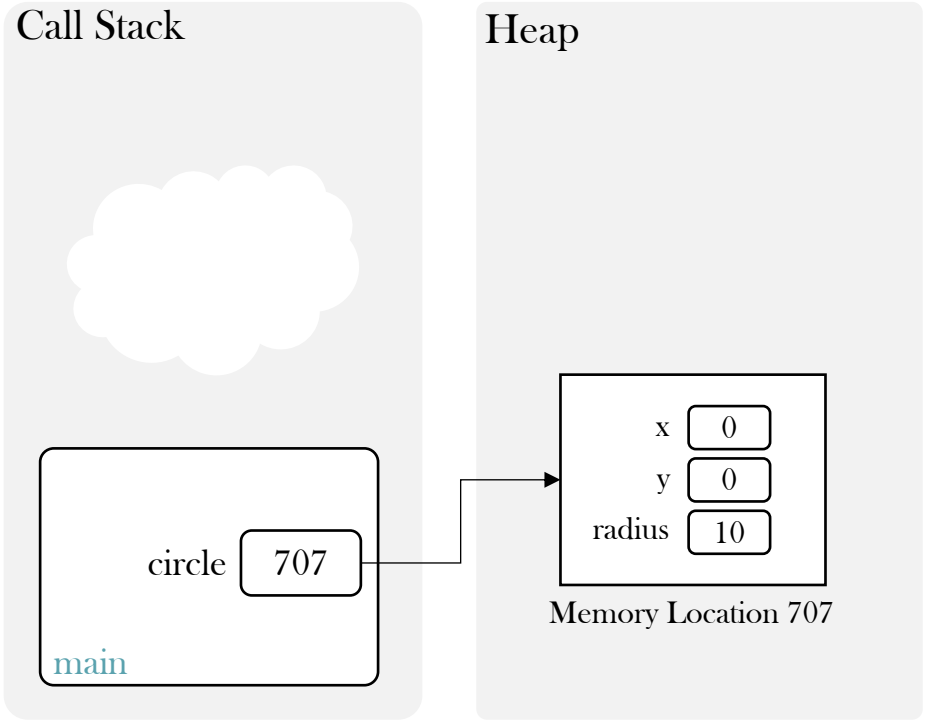


Program output

```
Radius: 10.0, x= 4.0, y= 5.0
```

Call Stack and Heap Status

```
1  public class AppCircle {
2    public static void main(String[] args) {
3
4      // create a circle, centered at x:4, y:5 with radius 10
5      Circle circle = new Circle(10,4,5);
6      circle.printInfo();
7      moveToOrigin(circle); // move the circle center to (x:0, y:0)
8      System.out.println("After method call");
9      circle.printInfo();
10   }
11
12   /**
13    * Moves the input circle center to (x:0,y:0)
14    * @param inputCircle Input circle object
15    */
16   private static void moveToOrigin(Circle inputCircle) {
17     inputCircle.setX(0);
18     inputCircle.setY(0);
19   }
20 }
```

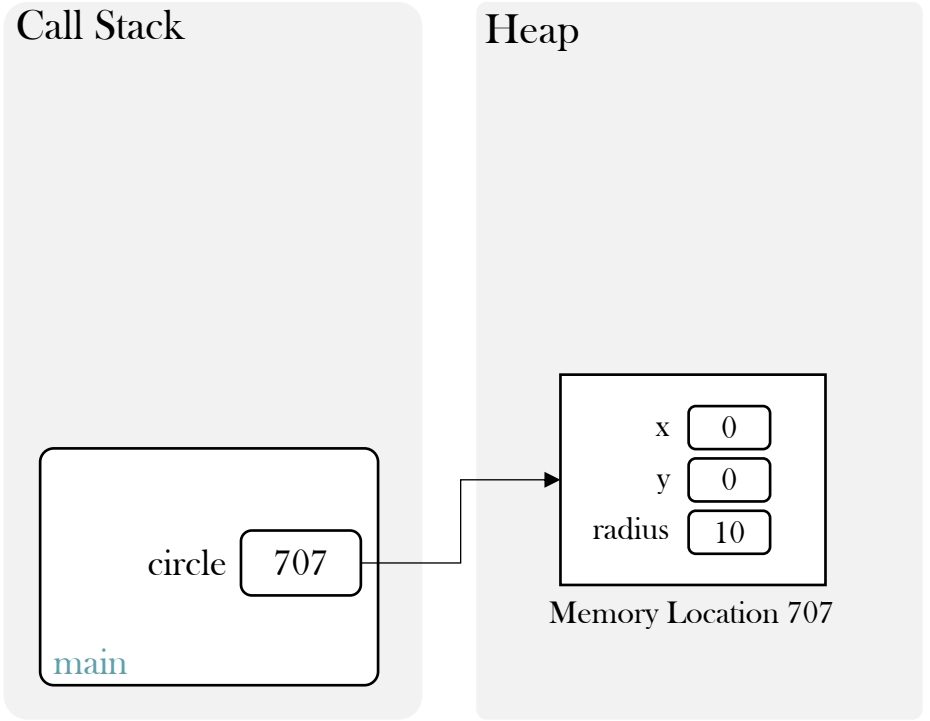


Program output

```
Radius: 10.0, x= 4.0, y= 5.0
After the method call
```

Call Stack and Heap Status

```
1  public class AppCircle {
2    public static void main(String[] args) {
3
4      // create a circle, centered at x:4, y:5 with radius 10
5      Circle circle = new Circle(10,4,5);
6      circle.printInfo();
7      moveToOrigin(circle); // move the circle center to (x:0, y:0)
8      System.out.println("After method call");
9      circle.printInfo();
10   }
11
12   /**
13    * Moves the input circle center to (x:0,y:0)
14    * @param inputCircle Input circle object
15    */
16   private static void moveToOrigin(Circle inputCircle) {
17     inputCircle.setX(0);
18     inputCircle.setY(0);
19   }
20 }
```



Program output

```
Radius: 10.0, x= 4.0, y= 5.0
After the method call
Radius: 10.0, x= 0.0, y= 0.0
```

Call Stack and Heap Status

```
1  public class AppCircle {
2    public static void main(String[] args) {
3
4      // create a circle, centered at x:4, y:5 with radius 10
5      Circle circle = new Circle(10,4,5);
6      circle.printInfo();
7      moveToOrigin(circle); // move the circle center to (x:0, y:0)
8      System.out.println("After method call");
9      circle.printInfo();
10   }
11
12   /**
13    * Moves the input circle center to (x:0,y:0)
14    * @param inputCircle Input circle object
15    */
16   private static void moveToOrigin(Circle inputCircle) {
17     inputCircle.setX(0);
18     inputCircle.setY(0);
19   }
20 }
```

Call Stack

Heap

Program output

```
Radius: 10.0, x= 4.0, y= 5.0
After the method call
Radius: 10.0, x= 0.0, y= 0.0
```

Returning Objects from Methods

Returning Objects from Methods

- A method can create an object and return it

```
public class App {
 public static void main(String[] args) {

  // create a big circle and store it in bigCircle variable
  Circle bigCircle = createBigCircle();
  bigCircle.printInfo();

 }

 /**
  * Creates a big circle with radius 100 at the origin
  * @return Big circle with radius 100, centered at the origin
  */
 private static Circle createBigCircle() {
  return new Circle(100,0,0);
 }
}
```

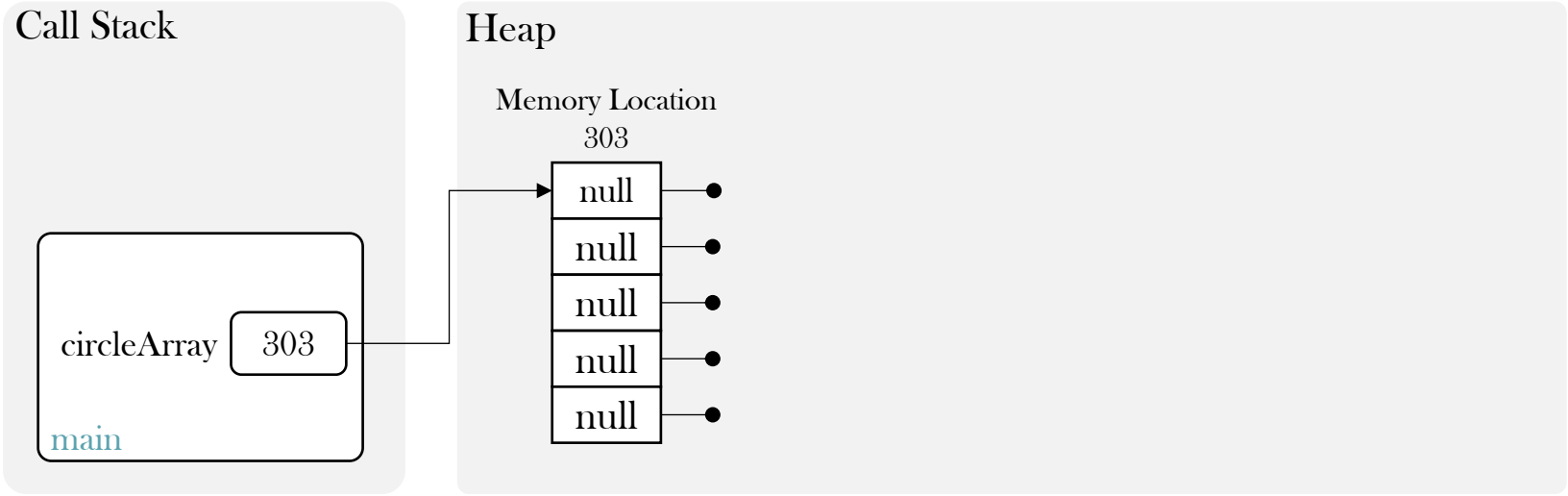
Arrays of Objects

Arrays of objects

- Arrays can hold objects. An array of objects is actually an array of reference variables
- For example, the following statement declares and creates an array of Circle objects:

```
Circle[] circleArray = new Circle[5];
```

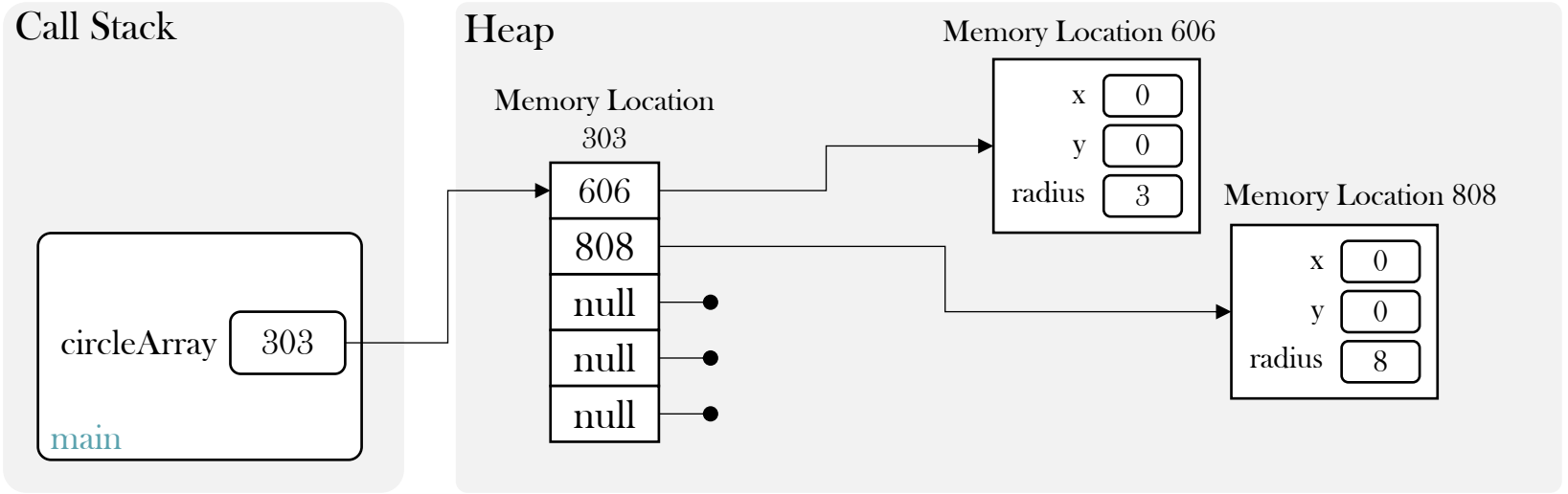
- Arrays contents are initially null:
  - Circle objects are not created yet. Only the empty array is created



Arrays of objects

- Let's create two circles in the array

```
Circle[] circleArray = new Circle[5];

circleArray[0] = new Circle(3);
circleArray[1] = new Circle(8);
```



Array of Objects

- Example: Create 10 circles with random radius and store them in an array

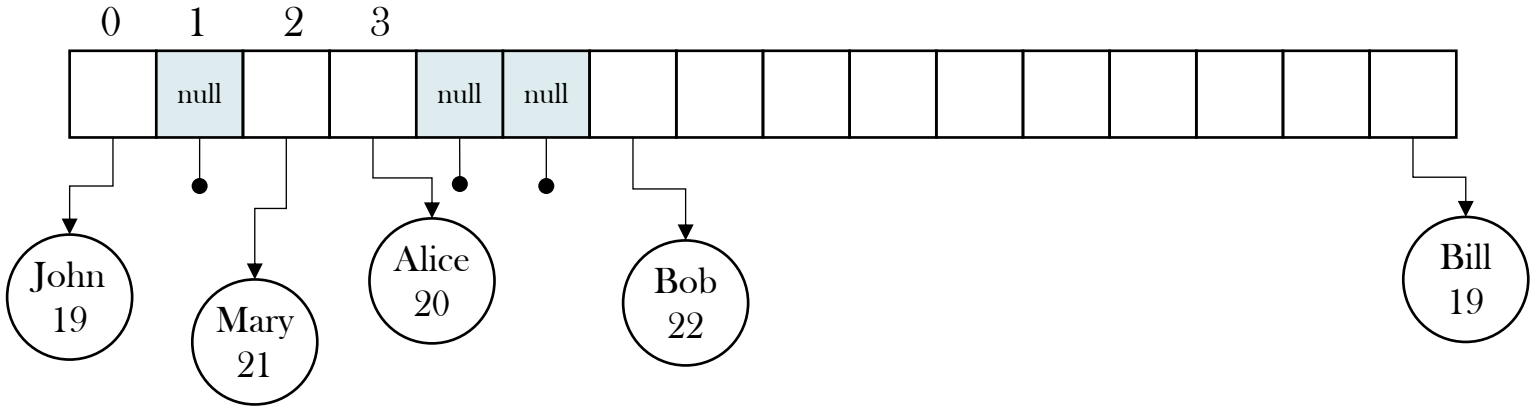
```
// create an array to store 10 circles
Circle[] circleArray = new Circle[10];
for (int i = 0; i < circleArray.length; i++) {

  // create a circle with random radius and place it into the array
  circleArray[i] = new Circle(Math.random()*10);

  // print circle info
  circleArray[i].printInfo();
}
```

- You can access members (data fields and class methods) of objects in an array using the following syntax:

```
circleArray[0].getArea(); // call the getArea method
circleArray[3].radius;    // get the radius of a circle in an array
```

Example: Array of Objects

- Store students who take COMP110 course in an array
  - Array length is 1000
  - There are 73 students: Some array elements are empty
- Print all student's information using for-each loop
  - Check if an array entry is null or not: If the array entry is null, do not print anything



Example: Array of Objects

```
public class App {
 public static void main(String[] args) {
  // create student array
  Student[] studentArray = new Student[1000];

  studentArray[0] = new Student("John",19);
  studentArray[2] = new Student("Mary",20);
  studentArray[3] = new Student("Alice",21);
  studentArray[6] = new Student("Bob",22);
  // continue adding other students
  studentArray[999] = new Student("Bill",19);

  // print all student's information using for-each loop
  for (Student s : studentArray)
   if (s != null) // if array element is null, do not print
    System.out.println(s); // Student class has toString method
   }
}
```

Program output

```
Student [name=John, age=19, department=COMP, grades=null, gpa=3.0]
Student [name=Mary, age=20, department=COMP, grades=null, gpa=3.0]
Student [name=Alice, age=21, department=COMP, grades=null, gpa=3.0]
Student [name=Bob, age=22, department=COMP, grades=null, gpa=3.0]
Student [name=Bill, age=19, department=COMP, grades=null, gpa=3.0]
```

Scope of Class Variables

Scope of Class Variables

- The scope of instance variables is the entire class, regardless of where the variables are declared
- A variable defined inside a method is referred to as a local variable

```
public class Circle {

  public double getArea() {
    double myPI = Math.PI; // myPI is a local variable
    return myPI * radius * radius;
  }

  private double radius; // you can define instance variable here, though not recommended
}
```

Scope of Class Variables

- You should declare a class variable only once
  - But you can declare the same variable name in methods many times, as a local variable
  - If a local variable has the same name as a class's variable, the local variable takes precedence and the class's variable is hidden
- Below, x is defined both as an instance variable and as a local variable

```
public class Ball {

  private int x = 1; // instance variable
  private int y = 1; // instance variable

  public void myMethod(double rate) {

    int x = 2; // x is a local variable
    System.out.println("Value of x: " + x);

  }
}
```

```
public class AppBall {
  public static void main(String[] args) {
    Ball b = new Ball();
    b.myMethod(0.2);
  }
}
```

Program output

```
Value of x: 2
```

Scope of Class Variables

- To avoid confusion and mistakes, do not use the names of instance variables as local variable names, except for method parameters

Anonymous Objects

Anonymous Objects

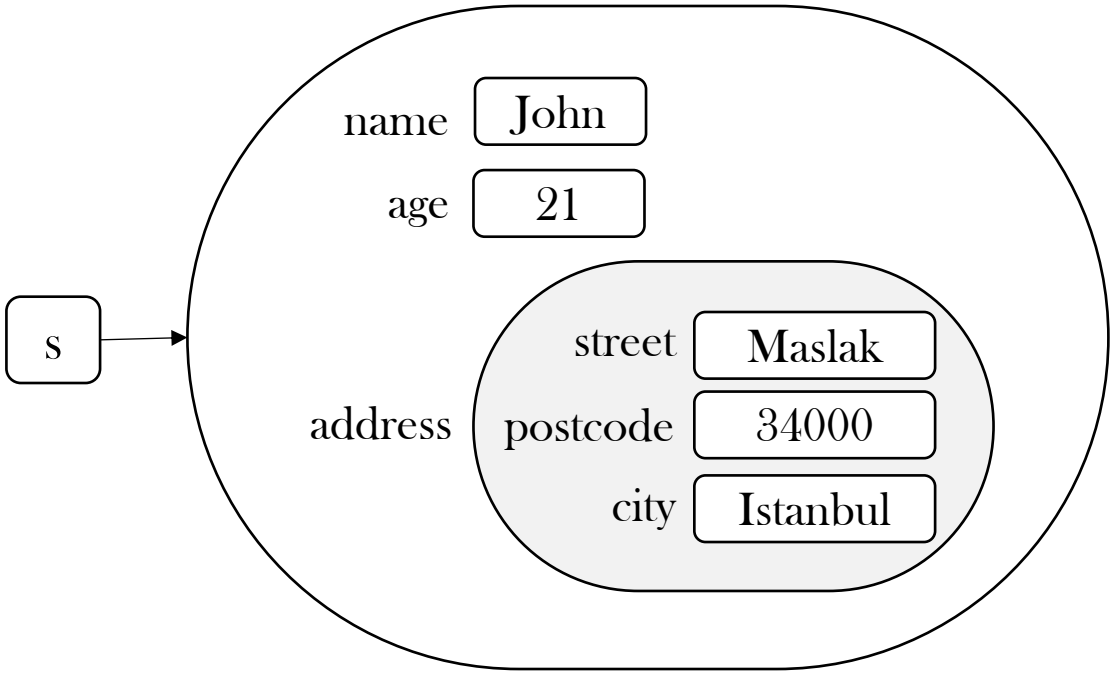
- Usually you create an object and assign it to a variable, then later you can use the variable
- Occasionally, an object does not need to be referenced later
- In this case, you can create an object without explicitly assigning it to a variable
- An object created in this way is known as an anonymous object

```
// object with a variable c
Circle c = new Circle(3);
System.out.println("Area of the circle: " + c.getArea());

// object without a variable (anonymous object)
System.out.println("Radius of a circle: " + new Circle(14).radius);
```

```
// method with an input object
plotCircle(c);
// method with an anonymous input object
plotCircle( new Circle(9) );
```

Object Data Fields

- Type of an instance variable can be a class
- Example:
  - Students have name, age and address
  - Data type of the address instance variable is Address class with street, postcode and city data fields



Object Data Fields

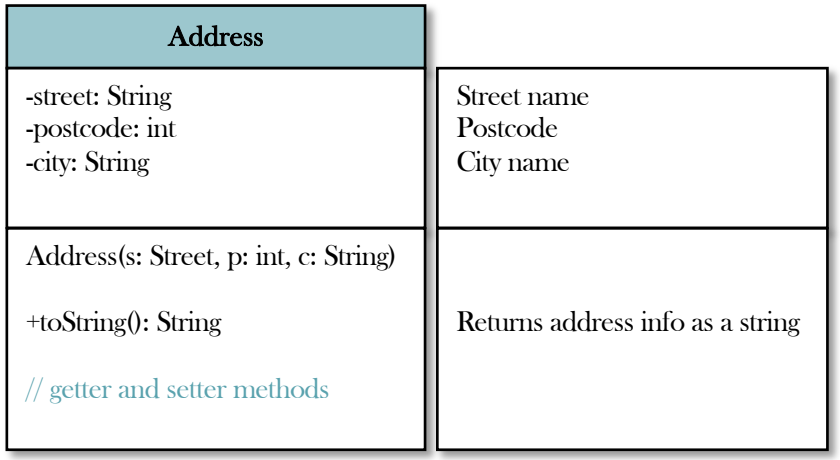
- Type of an instance variable can be a class
- Example:
  - Students have name, age and address
  - Address is a class with street, postcode and city data fields (See the UML class diagram below)

Student	
+name: String +age: int -address: Address	Student's name Age Student's grades
// constructors +setAddress(a: Address): void +getAddress(): Address +toString(): String	 Sets the address of student Gets the address of student Returns student info as a string

Address	
-street: String -postcode: int -city: String	Street name Postcode City name
Address(s: Street, p: int, c: String) +toString(): String // getter and setter methods	 Returns address info as a string

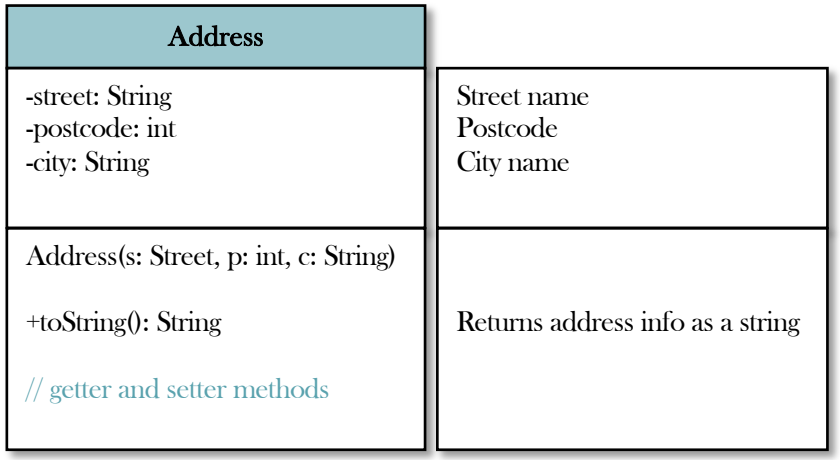
Address Class – Part 1/2

```
/**
 * Address class stores address information which contains
 * street name, postcode and city name
 *
 * @author BG
 */
public class Address {

  private String street;
  private int postcode;
  private String city;

  public Address(String street, int postcode, String city) {
    this.street = street;
    this.postcode = postcode;
    this.city = city;
  }

  @Override
  public String toString() {
    return "Address [street=" + street + ", postcode=" + postcode + ", city=" + city
+ "]";
  }
  // code continues
}
```



Address Class – Part 2/2

```
public class Address {

  // code continues from here
  public String getStreet() {
    return street;
  }
  public void setStreet(String street) {
    this.street = street;
  }
  public int getPostcode() {
    return postcode;
  }
  public void setPostcode(int postcode) {
    this.postcode = postcode;
  }
  public String getCity() {
    return city;
  }
  public void setCity(String city) {
    this.city = city;
  }
}
```



Student Class

```
public class Student {

  public String name;
  public int age;
  private Address address;

  // other data fields

  public Student(String name, int age, String department, int[] grades, double gpa, Address address) {
    this.name = name;
    this.age = age;
    this.address = address;
    // assign other data fields
  }

  public Student(String name, int age) {
    this(name, age, "COMP", null, 3.10, null);
  }

  public Address getAddress() { return address; }

  public void setAddress(Address address) { this.address = address; }

  @Override
  public String toString() {
    return "Student [name=" + name + ", age=" + age + ", department=" + department + ", grades="
        + Arrays.toString(grades) + ", gpa=" + gpa + ", address=" + address + "]";
  }
  // other class methods
}
```

Student	
+name: String +age: int -address: Address	Student's name Age Student's grades
// constructors +setAddress(a: Address): void +getAddress(): Address +toString(): String	 Sets the address of student Gets the address of student Returns student info as a string

Main Application for Students

```
public class App {
 public static void main(String[] args) {

  // create student John
  Student s = new Student("John",19);

  // create an address for John and assign to him
  Address a = new Address("Maslak",34000,"Istanbul");
  s.setAddress(a);
  System.out.println(s);

  // create student Mary
  Student p = new Student("Mary",21);
  // create an anonymous address object for Mary
  p.setAddress( new Address("Besiktas",31000,"Istanbul") );
  System.out.println(p);
 }
}
```

Program output

```
Student [name=John, age=19, department=COMP, grades=null, gpa=3.1, address=Address [street=Maslak, postcode=34000, city=Istanbul]]
Student [name=Mary, age=21, department=COMP, grades=null, gpa=3.1, address=Address [street=Besiktas, postcode=31000, city=Istanbul]]
```

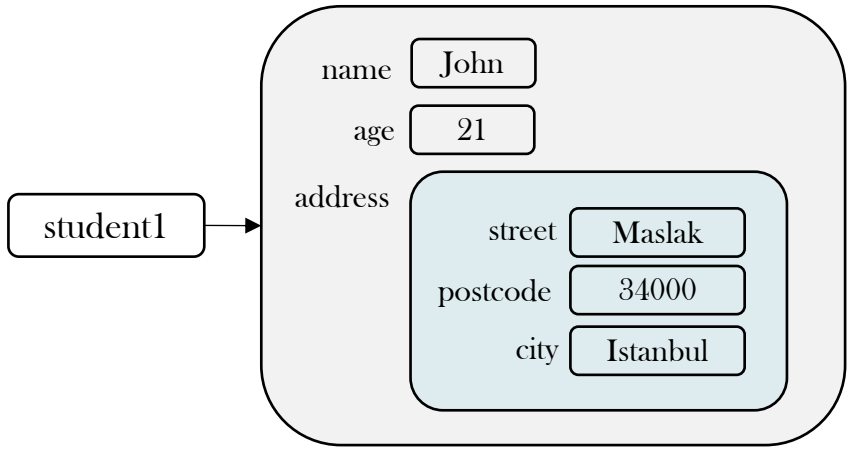
Accessing Data Fields

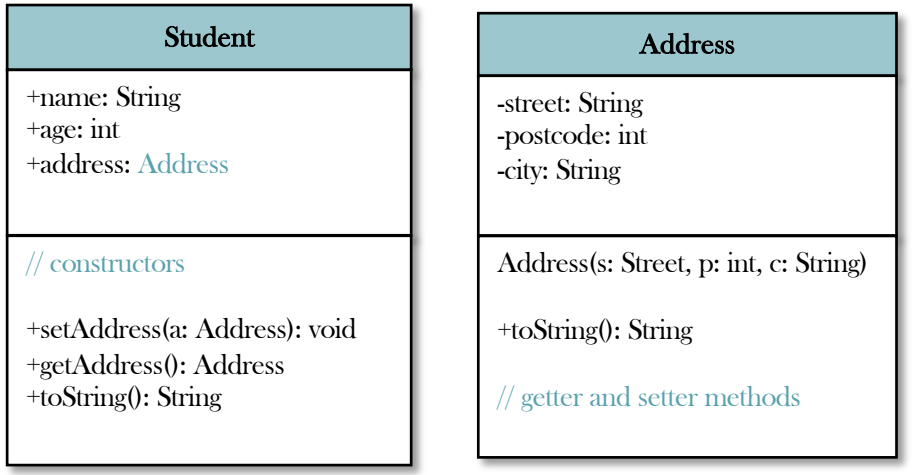
- You can access the data fields and methods of an instance variable which is an object type using the dot (.) operator
- Examples:
  Accessing the street data field of a student (assume all data fields are public)

```
String var = student1.address.street;
```

  Accessing the getPostcode method of a student (assume all methods are public)

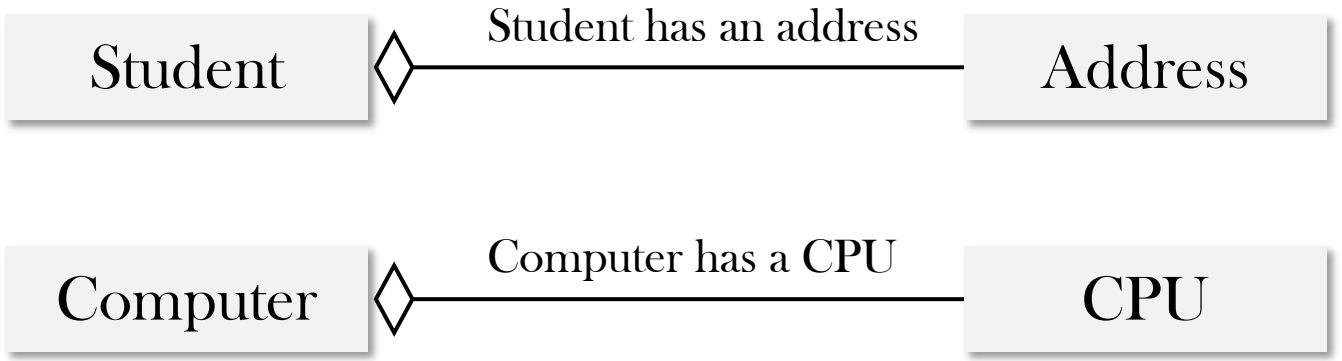
```
String var = student1.address.getPostcode();
```





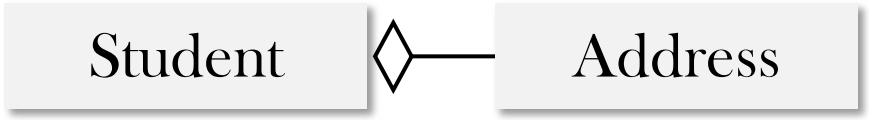
Aggregation Relationship between Classes

- Has-a relationship between classes:
  - Student has an address
  - Computer has a CPU
- has-a relationships between classes are refered to as aggregation relationships
- UML notation for aggregation relationship is empty diamond symbol
  - Diamond symbol is next to the aggregating class, e.g., Student

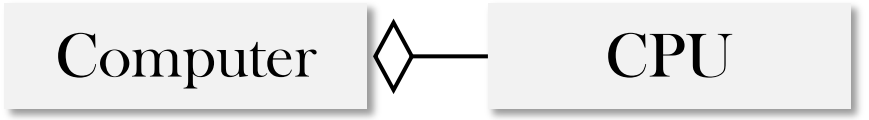


Aggregation Relationship between Classes

- Has-a relationship is usually modeled using data fields




```
public class Student {

 private Address myAddress;
 // other data fields

}
```




```
public class Computer {

 private CPU cpuModel;
 // other data fields

}
```

Static Variables and Methods

Static Variables

- The data field radius in the circle class is known as an instance variable
- An instance variable is tied to a specific instance of the class; it is not shared among objects of the same class
- For example, suppose that you create the following objects:

```
Circle circle1 = new Circle(2);
Circle circle2 = new Circle(5);
```

- The radius in circle1 is independent of the radius in circle2 and is stored in a different memory location
- Changes made to circle1's radius do not affect circle2's radius, and vice versa

Static Variables and Methods

- If you want all the instances of a class to share data, use static variables
- Static variables store values for the variables in a common memory location
- Because of this common location, if one object changes the value of a static variable, all objects of the same class are affected
- Java supports static methods: Static methods can be called without creating an instance of the class

Declaring Static Variables and Methods

- To declare a static variable or define a static method, put the modifier static in the variable or method declaration

```
public class Product {

 public String brand;
 public double price;
 public static int numberOfProducts = 0; // static variable

 Product() { this("Default brand name", 0); }

 Product(String inputBrand, double inputPrice) {
  brand = inputBrand;
  price = inputPrice;
  numberOfProducts++; // increase the static variable
 }

 // static method
 public static int getNumberOfProducts() {
  return numberOfProducts;
 }
}
```

Accessing Static Members

- Instance methods and instance variables belong to instances and can be used only after the instances are created:
  - They are accessed via a reference variable such as myCircle.getArea();
- Static methods and static data can be accessed from an object reference variable or from their class name
  - Use ClassName.methodName() to invoke a static method, such as Circle.getNumberOfCircles()
  - Use ClassName.staticVariable to access a static variable, such as Circle.numberOfCircles
- Using class name to access static members improves readability because this makes static methods and data easy to spot
- It is not recommended to use object reference variables to access static members

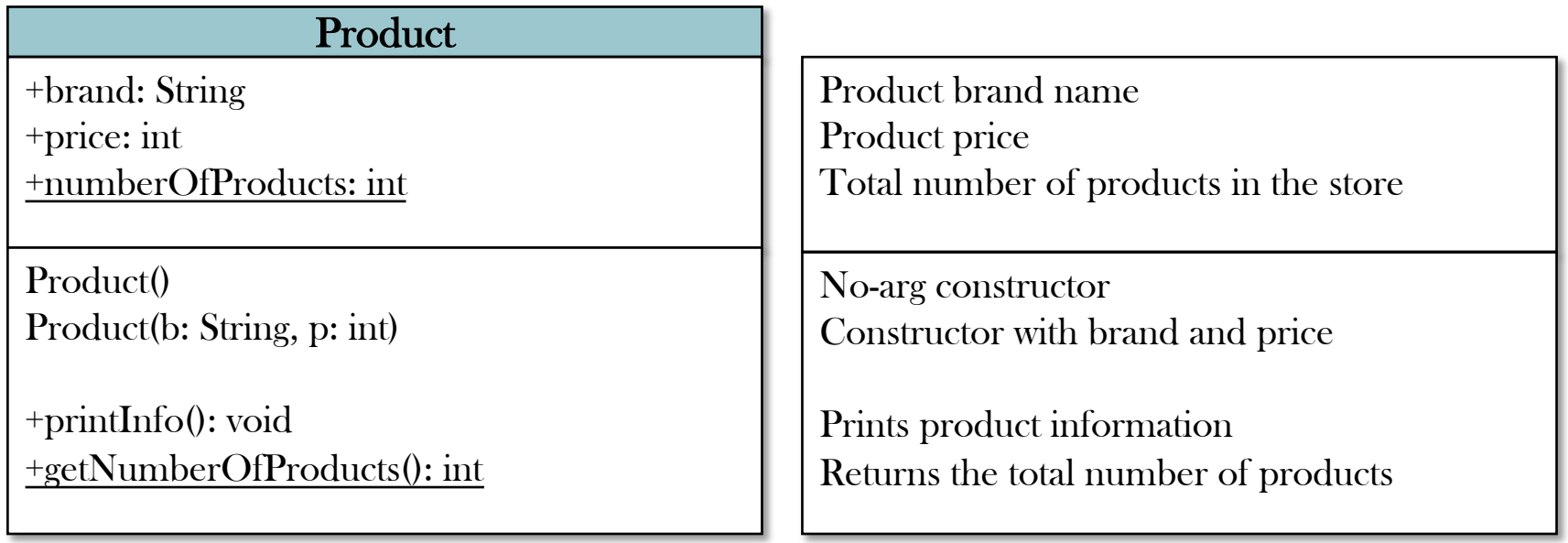
Accessing Static Members

- Use class name and dot (.) operator to access static variables and methods
- Since static members do not belong to objects, we do not use object reference variables to access them
  - `Product.getNumberOfProducts() // correct: static variable belongs to class`
  - `p.getNumberOfProducts() // not recommended`

```
public class AppProduct {
 public static void main(String[] args) {

   Product p = new Product();
   p.brand = "Adidas";
   p.price = 200.0;

   Product s = new Product();

   System.out.println(Product.numberOfProducts);
   System.out.println(Product.getNumberOfProducts());

 }
}
```

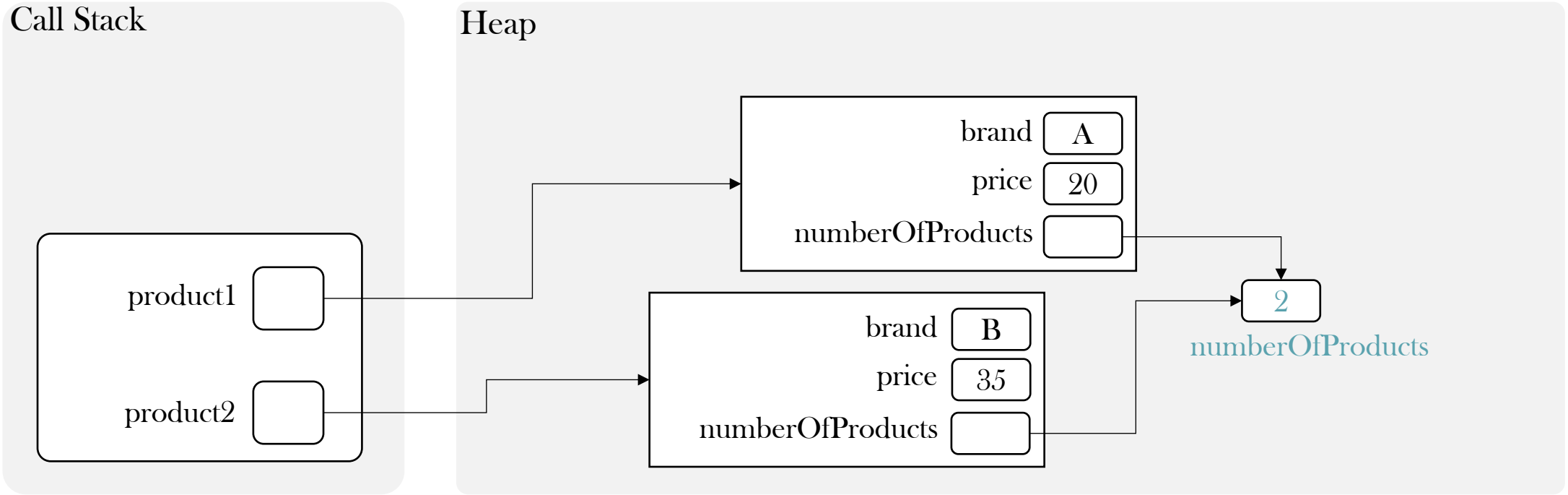
UML Notation for Static Variables

- Static variables and methods are underlined in UML class diagrams
- In the example below, data field numberOfProduct and class method getNumberOfProducts are static



Memory Map for Call Stack Heap for Static

- Static variables are shared by all objects: So they are stored in a single -separate- shared location, outside of the objects



Static Methods Can Not Access Instance Members

- A static method can invoke a static method and access a static data field
- A static method cannot invoke an instance method or access an instance data field, since static methods and static data fields don't belong to an object
- Instance methods can access static variables and methods

```
public class Product {

 public String brand;
 public double price;
 public static int numberOfProducts = 0;

 public static int getNumberOfProducts() {
  System.out.println("This is a static method");
  price = 100; // this is wrong: static method cannot access instance variable price
  return numberOfProducts;
 }
}
```

Design Guide for Static/Instance Members

- How do you decide whether a variable or a method should be instance or static?
  - A variable or a method that is dependent on a specific instance of the class should be instance
  - A variable or a method that is not dependent on a specific instance of the class should be static
- For example
  - Every circle has its own radius: so radius is an instance variable
  - getArea method is dependent on a specific circle: so it an instance method
- Example for static methods and variables
  - None of the methods in the Math class, such as random, pow, sin, and cos, is dependent on a specific instance. Therefore, these methods are static methods
  - Math class has PI variable (Math.PI) and it does not belong to any object. It is defined as static

Constants as Local Variables

- The value of a variable may change during the execution of a program, but a constant represents permanent data that never changes
  - Syntax for declaring a constant: final datatype CONSTANTNAME = value;
  - A constant must be declared and initialized in the same statement
  - The word final is a Java keyword for declaring a constant
  - By convention, all letters in a constant are in uppercase
- For example, you can declare PI as a constant: final double PI = 3.14159;

```
public class App {
 public static void main(String[] args) {

   final double PI = 3.1415;
   System.out.println(PI);
   PI = 5.0; // error: final local variable cannot be assigned

 }
}
```

Constants as Data Fields

- Constants in a class are shared by all objects of the class
- Thus, constants should be declared as final static
- If a constant is a data field of a class, use static keyword

```
public class Circle {

 private double radius;
 private double x,y;

 private final static double GRAVITY = 9.8;

 public void dropCircle() {
  System.out.println(GRAVITY);
  GRAVITY = 12.0; // error: final data field cannot be assigned
 }
}
```