

Java Programming

Arthur Hoskey, Ph.D.
Farmingdale State College
Computer Systems Department

- Classes (continued)

Today's Lecture

- Private member
 - Only visible from inside the object
 - Cannot be seen from "outside"
- Public members
 - Visible from the outside
- Now look at the class definition again:
 - Look for the public members
 - Look for the private members
- For example...

REVIEW - Access Modifiers

```
public class Car
{
    // Attributes
    private int year;
    private int speed;
    private String color;

    // Behaviors
    public void Accelerate() {
        speed = speed + 10;
    }
    public void Decelerate() {
        speed = speed - 10;
    }
}
```

REVIEW - Sample Class Definition

- **private** keyword
 - **Used for most instance variables.**
 - private variables and methods are accessible only to methods of the class in which they are declared.
 - Declaring instance variables private is known as "**data hiding**".
- **public** keyword
 - **Used for most methods.**
 - Public methods are accessible outside the class.

REVIEW - Access Modifiers

- If the private members cannot be seen from the outside the class then how do we change them?

REVIEW - Access Modifiers

- **Use get/set methods to change private member variables.**
- private instance variables
 - Cannot be accessed directly by clients of the object.
 - Use **set** methods to **change the value.**
 - Use **get** methods to **retrieve the value.**

REVIEW - Get and Set Methods

```
public class Car
{
    // Attributes
    private int year;
    private int speed;
    private String color;

    // Behaviors
    public int GetYear() { return year; }
    public int GetSpeed() { return speed; }
    public String GetColor() { return color; }

    public void SetYear(int newYear) { year = newYear; }
    public void SetSpeed(int newSpeed) { speed = newSpeed; }
    public void SetColor(String newColor) { color = newColor; }

    // Accelerate and Decelerate not shown
}
```

REVIEW - Get and Set Methods

- **Local Variables** - Declared in the body of method. Can only be used within that method.
- **Instance Variables** - Declared in a class declaration but not in a method. Each object of the class has a separate instance of the variable.

```
public class MyClass {  
    int x;  
  
    public void myMethod {  
        int y;  
        y = 10;  
        x = 20;  
    }  
  
    public void otherMethod() {  
        x = 20;  
  
        y = 30;  
    }  
}
```

x is an instance variable (accessible from all methods of the class)

y is a local variable for myMethod (only accessible from inside myMethod)

x can be used in both of these places because member methods have access to all member variables

Y is local to myMethod so it CANNOT be used here (y is out of scope)

REVIEW - Scope of variables

```
public class Car
{
    // Attributes
    private int year;
    private int speed;
    private String color;
```


```
    // Behaviors
    Get and set methods not shown
```

```
    public void Accelerate() {
        speed = speed + 10;
    }
    public void Decelerate() {
        speed = speed - 10;
    }
}
```

Variable Resolution In Member Method

1. First, look for a local declaration of the variable. If found then use it.
2. Second, look for the variable in the class scope.

**No local speed variable
so it uses the member
variable speed**



**Are you allowed to declare both
a local variable and a class-level
variable with the same name???**

Variable Resolution

- What will memory allocations (in RAM) look like for the following code?

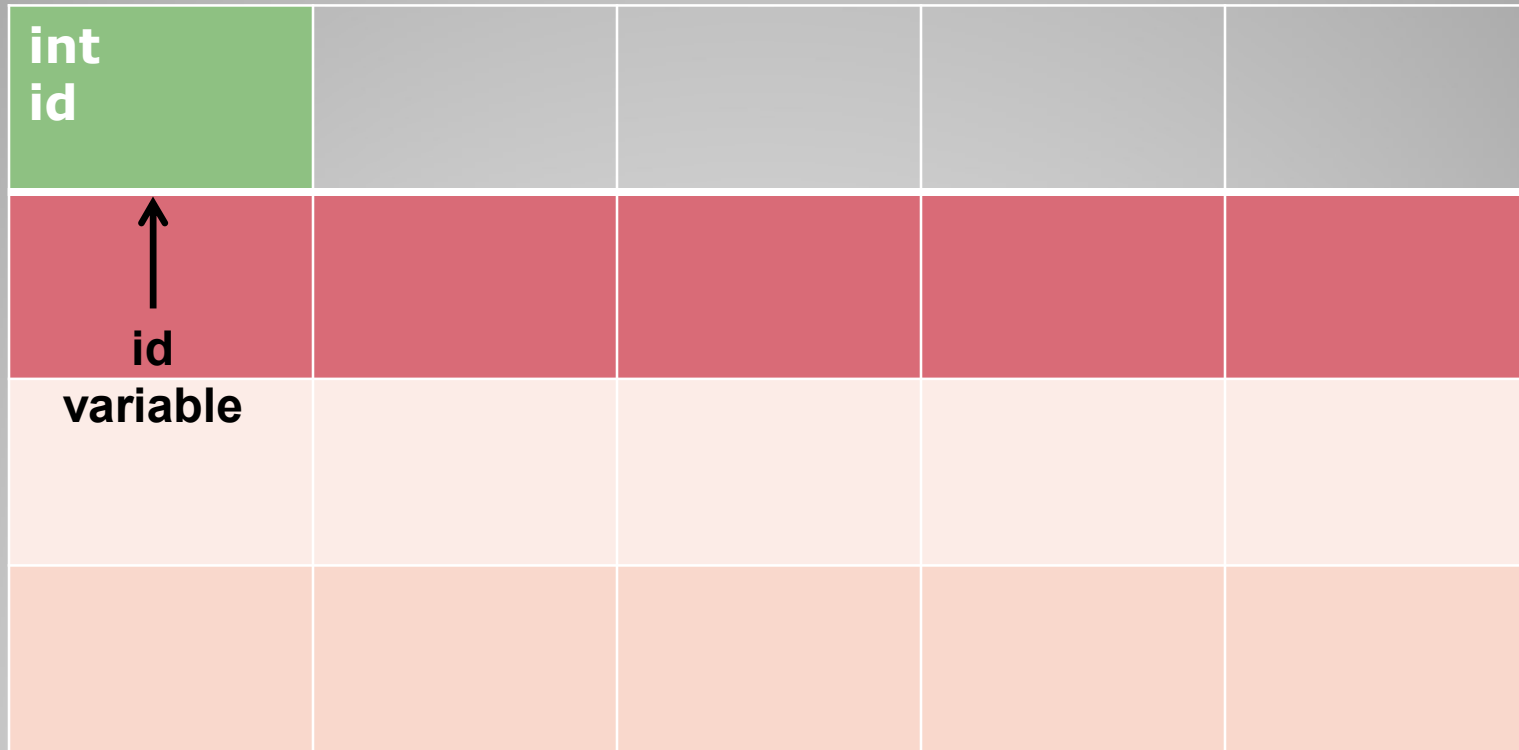
```
public static void main(String []args) {  
    int id;  
}
```

**main() is the starting
point for all Java
programs**

- Declared one int type variable.

Classes In Memory

Computer Memory (RAM)



Showing a simplified view of
memory here but will go into more
detail later in the semester

**int takes up 4
bytes in
memory**

Variables In Memory (RAM)

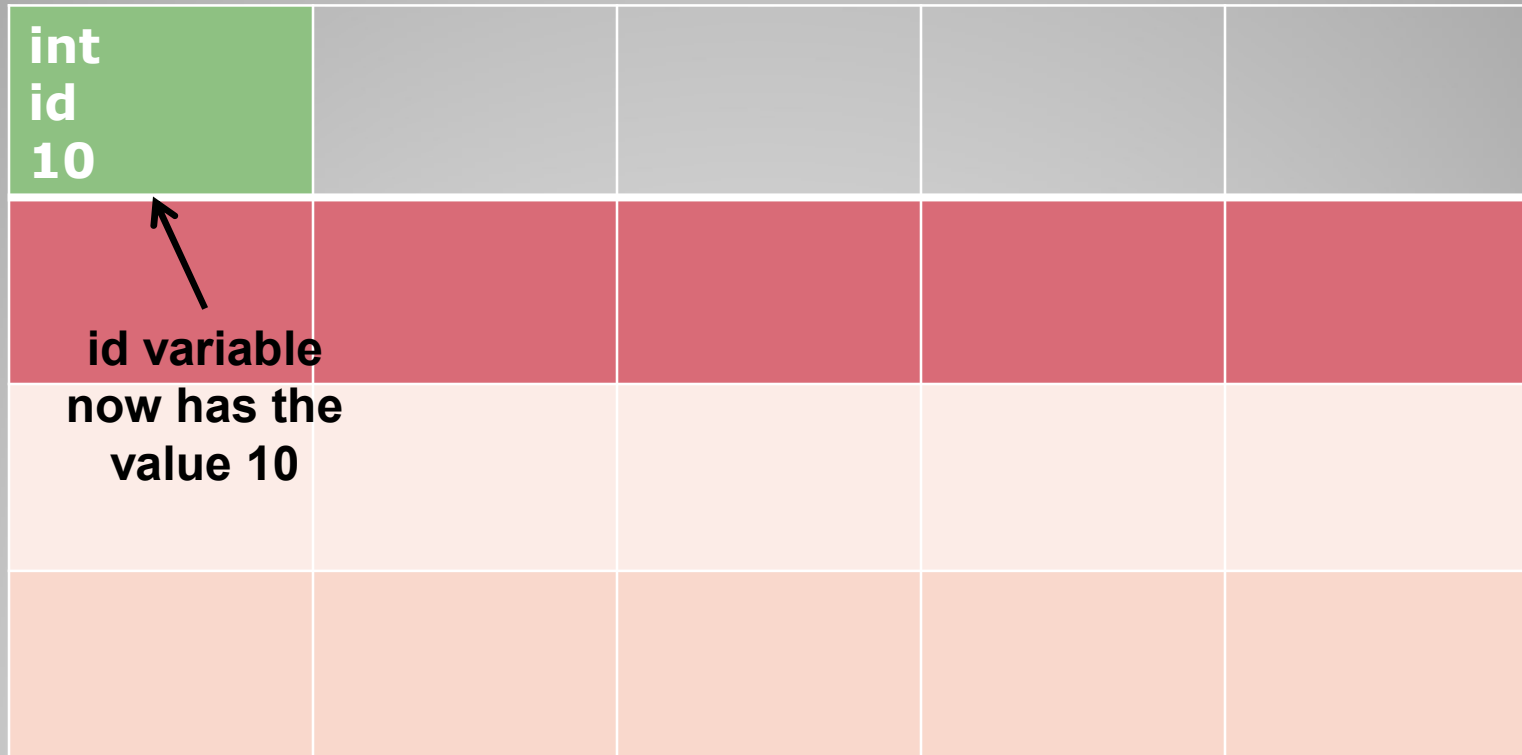
- What will memory allocations (in RAM) look like for the following code?

```
public static void main(String []args) {  
    int id;  
    id = 10;  
}
```

**Declare variable and
assign value**

Classes In Memory

Computer Memory (RAM)



Variables In Memory (RAM)

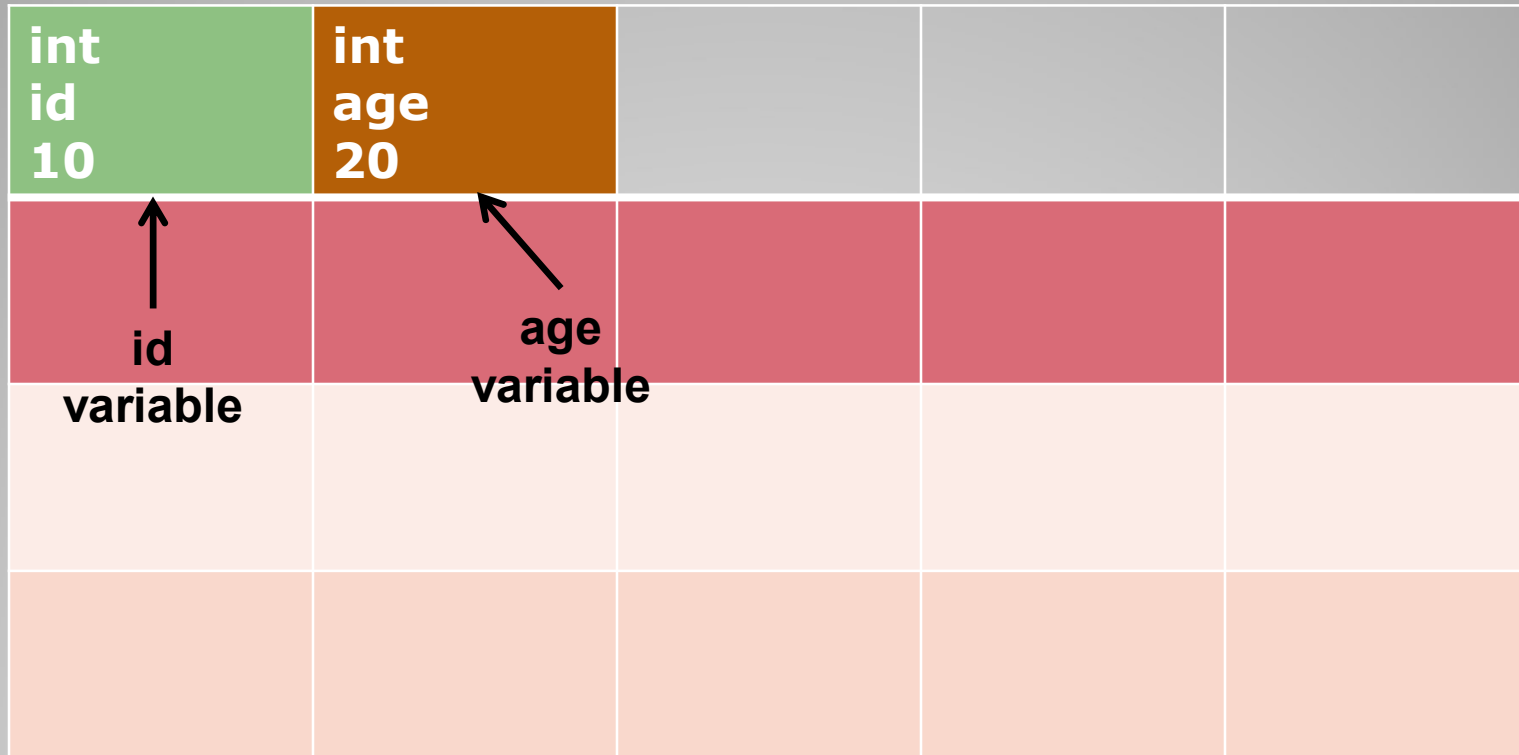
- What will memory allocations look like for the following code?

```
public static void main(String []args) {  
    int id = 10;        // Declare and initialize  
    int age = 20;      // Declare and initialize  
}
```

- Declared two int type variables.

Classes In Memory

Computer Memory (RAM)



If you make a change to the id variable will it effect the age variable?

Classes In Memory

- Assume the previous definition of the Car class.
- What will memory allocations (in RAM) look like for the following code?

```
public static void main(String []args) {  
    int id = 10;  
    int age = 20;  
    String d = "Yanks";
```

```
    Car myCar; // Declaring a variable of type Car
```

```
    myCar = new Car(); // Call new to create instance  
}
```

Classes In Memory **Is there a memory leak??**

Computer Memory (RAM)

int id 10	int age 20	int year	int speed	String color
String d "Yanks"				

myCar
(contains 3 member variables, 2 int and 1 String)

This is a simplified view of what is happening
with classes in memory

Classes In Memory

- What will memory look like if we declare and create another instance of the car class?

Classes In Memory

- Assume the previous definition of the Car class.
- What will memory allocations (in RAM) look like for the following code:

```
public static void main(String[] args)
{
    int id = 10;
    int age = 20;
    String d = "Yanks";
    Car myCar;
    Car anotherCar; // Different car object

    myCar = new Car();
    anotherCar= new Car();
}
```

Classes In Memory

Computer Memory (RAM)

int id 10	int age 20	int year	int speed	String color
String d "Yanks"	int year	int speed	String color	

anotherCar
variable

myCar
variable

**EVERY instance of Car has its own
full set of the member variables!!!**

Classes In Memory

- How do we use the Car class in code?

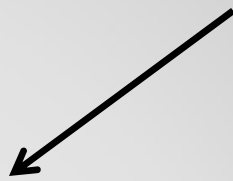
Access Modifiers

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

    myCar = new Car();

    myCar.SetSpeed(10);
    myCar.SetYear(2020);
    myCar.SetColor("black");
}
```

**What does memory
look like AFTER
running this line?**



Classes In Memory

Computer Memory (RAM)

int year 2020	int speed 10	String color "black"		

↑
myCar
variable

**If you change the year member variable
will it effect the speed member variable?**

Classes In Memory


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

    myCar = new Car();

    SetSpeed(10);
    myCar.SetYear(2020);
    myCar.SetColor("black");
}
```

**What is wrong
with this
code???**

Classes In Memory

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

```
{
```

```
    Car myCar;
```

What is wrong with this
code???

```
    myCar = new Car();
```

ANSWER
You must call the member
method with respect to an
instance variable

```
        SetSpeed(10); // Incorrect
```

```
    myCar.SetYear(2020);
```

```
    myCar.SetColor("black");
```

```
}
```

Classes In Memory

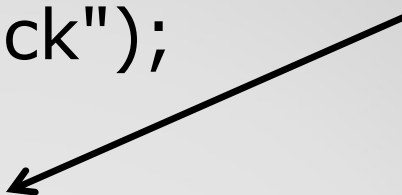
```
public static void main(String[] args)
{
    Car myCar;

    myCar = new Car();

    myCar.SetSpeed(10);
    myCar.SetYear(2020);
    myCar.SetColor("black");

    myCar.Accelerate();
}
```

**What does
memory look like
AFTER running
this line?**



Classes In Memory

Computer Memory (RAM)

int year 2020	int speed 20	String color "black"		

↑
myCar
variable

The speed member variable now has the value 20

Classes In Memory

```
public static void main(String[] args)
{
    Car myCar, anotherCar;

    myCar = new Car();
    anotherCar = new Car();

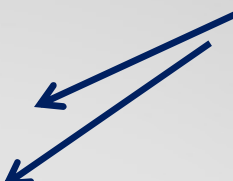
    myCar.SetSpeed(10);
    myCar.SetYear(2020);
    myCar.SetColor("black");

    anotherCar.SetSpeed(20);
    anotherCar.SetYear(2022);
    anotherCar.SetColor("red");
}
```


Must call new for every instance



Must call member methods with respect to a given instance

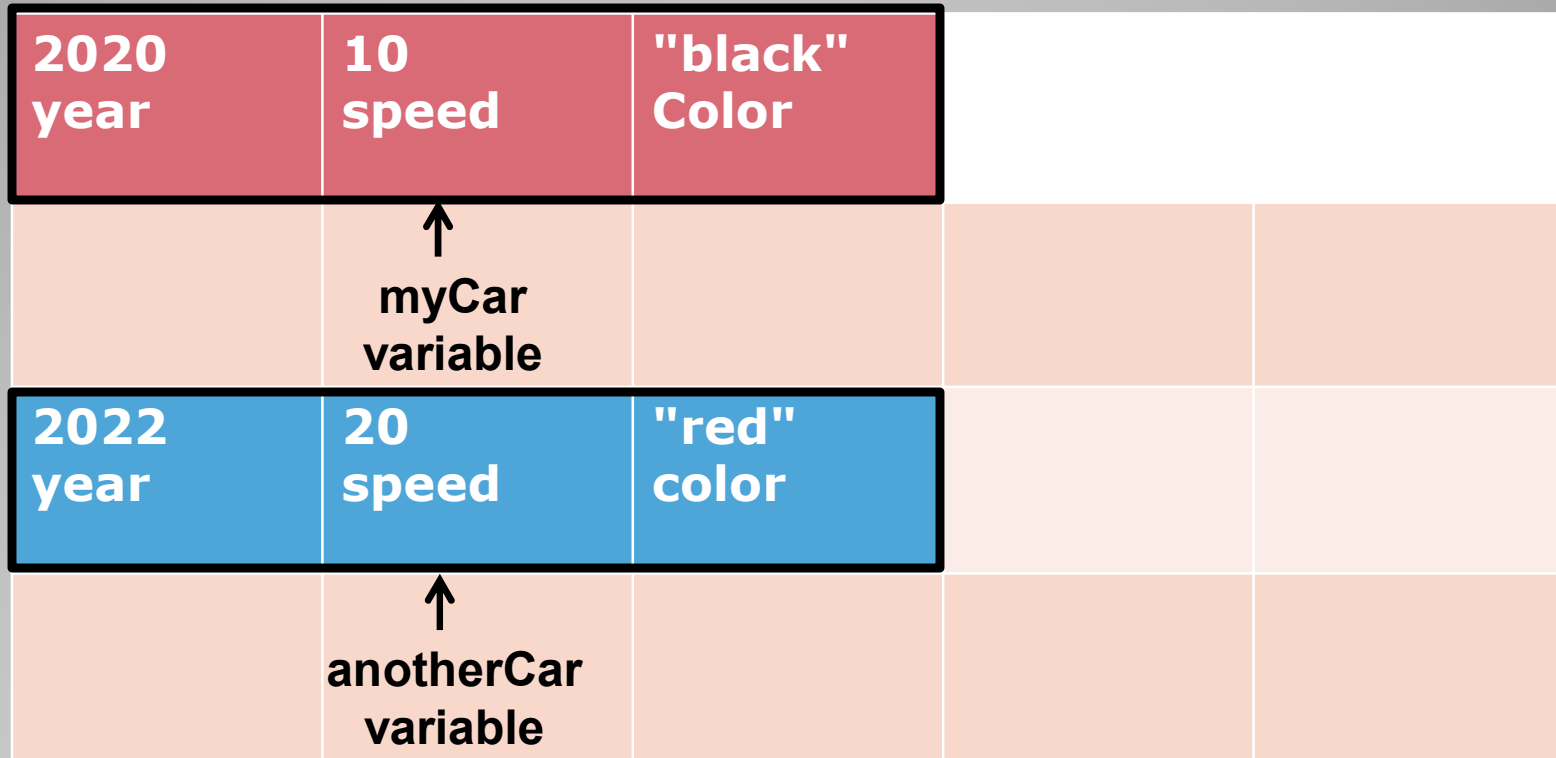


What does memory look like AFTER running this line?



Classes In Memory

Computer Memory (RAM)



Each instance of the Car class has its own section of memory.

Classes In Memory

- If we call the Accelerate() method on the anotherCar instance what will memory look like?
- For example...

Calling a Method

```
public static void main(String[] args)
{
    Car myCar, anotherCar;

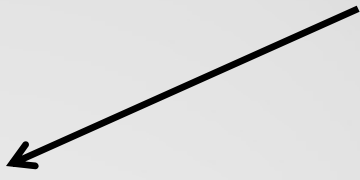
    myCar = new Car();
    anotherCar = new Car();

    myCar.SetSpeed(10);
    myCar.SetYear(2020);
    myCar.SetColor("black");

    anotherCar.SetSpeed(20);
    anotherCar.SetYear(2022);
    anotherCar.SetColor("red");

    anotherCar.Accelerate();
}
```

**Call Accelerate()
on the
anotherCar
instance**



Calling a Method

Computer Memory (RAM)

2020 year	10 speed	"black" Color		
	↑ myCar variable		After calling Accelerate() on anotherCar the speed of anotherCar changed but <u>NOT</u> the speed of myCar	
2022 year	30 speed	"red" color		
	↑ anotherCar variable			

If you make a change to one instance it does NOT effect any other instance.

Calling A Method

- How do we initialize an object?
- A special method called a constructor is used to initialize an instance of an object.
- The constructor gets called when the "new" method runs.
- For example...

Constructors

Car class contains a constructor

```
public class Car
{
    // Attributes
    private int year;
    private int speed;
    private String color;

    // Behaviors
    Get/Set and Accelerate and Decelerate methods not shown

    // Default Constructor – Takes no parameters
    public Car() {
        year = 2021;
        speed = 0;
        color = "Red";
    }
}
```

Constructors

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

    // New instance using a default
    // constructor
    myCar = new Car();
}
```

**This call to new will
call the default
constructor**



Constructors

Computer Memory (RAM)

int year 2021	int speed 0	String color "red"		

↑
myCar
variable

Memory After Default Constructor Runs

- Default constructor
- Takes no parameters
- Assigns starting values to attributes
- If you do not define **any** constructors then a default constructor is created automatically by the compiler behind the scenes.
- This automatically created default constructor initializes all attributes to their default values (for example, `int` \rightarrow 0).

Constructors

```
public class Car  
{
```

```
    // Attributes  
    private int year;  
    private int speed;  
    private String color;
```

```
    // Behaviors  
    // Get/Set go here...  
    // Accelerate goes here...  
    // Decelerate goes here...
```

```
}
```

**This Car class does NOT
contain a
constructor**

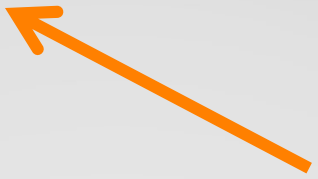
**There are NO
consturctors defined
so the compiler will
create a default one
automatically**

Constructors

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

    // New instance
    myCar = new Car();
}
```

**New calls the
automatically
generated default
constructor for
the class**



Constructors


- Constructors with parameters
- You can initialize attributes to any value by passing in data to the constructor.
- Use a parameter for every value that you want to be able to initialize from outside the class.
- For example...

Constructors

```
public class Car
{
    // Attributes
    private int year;
    private int speed;
    private String color;

    // Behaviors
    Get/Set and Accelerate and Decelerate methods not shown

    // Constructor
    public Car(int newYear, int newSpeed, String newColor) {
        year = newYear;
        speed = newSpeed;
        color = newColor;
    }
}
```



Sets all the values

Constructors

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

    // New instance using a constructor
    myCar = new Car(2022, 15, "blue");
}
```



New calls the constructor for the class. Parameters are passed in to the constructor like a method call.

Constructors

Computer Memory (RAM)

int year 2022	int speed 15	String color "blue"		

↑
myCar
variable

Memory After Constructor Runs

- Java data types are divided into two major categories:
Primitive and Reference
- Primitive types are the following:
boolean, byte, char, short, int, long, float, double
- Class instances are reference types
- You must call the “new” operator to instantiate a reference type

Data Types

- There are reference versions of the primitive types (wrapper classes).
- These can be used when you need to put a primitive value where a reference value is required.

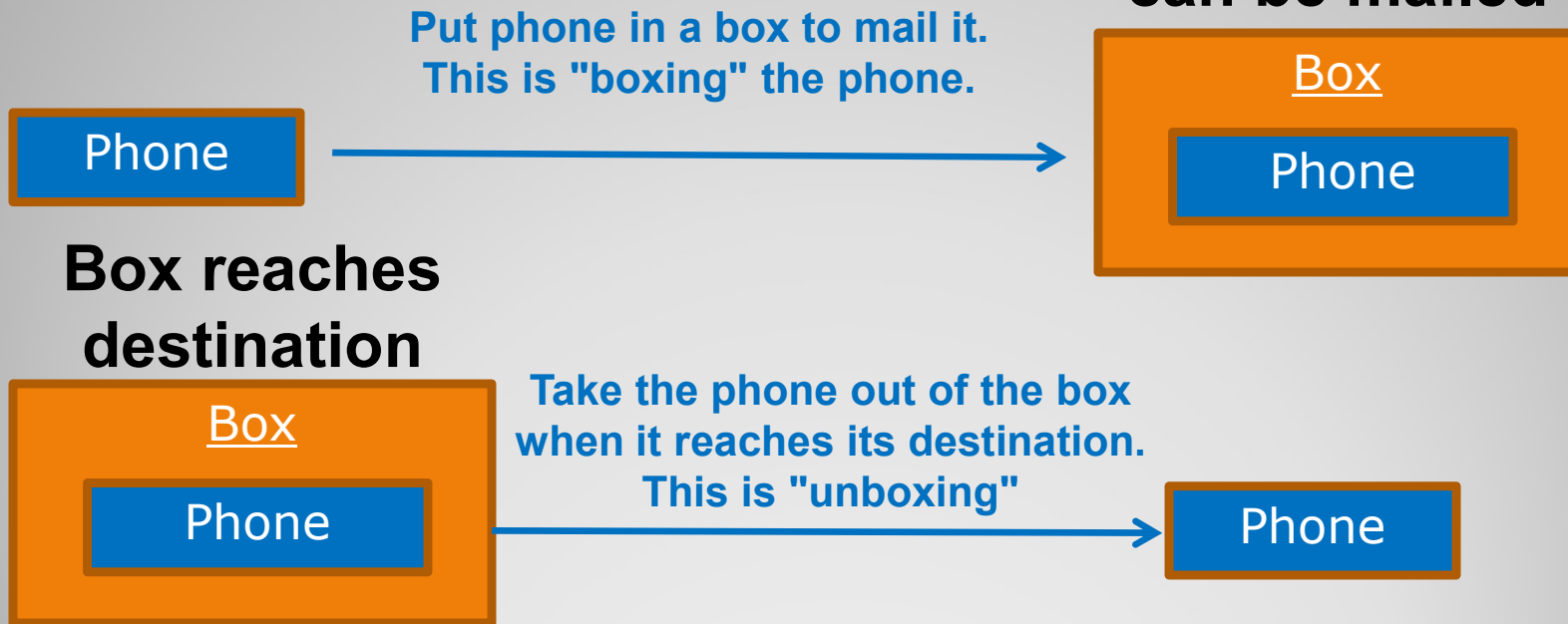
Primitive Type	Wrapper Class for Primitive Type
int	Integer
double	Double
char	Character
byte	Byte
boolean	Boolean
short	Short
long	Long
float	Float

Wrapper Data Types

Mailing a Phone

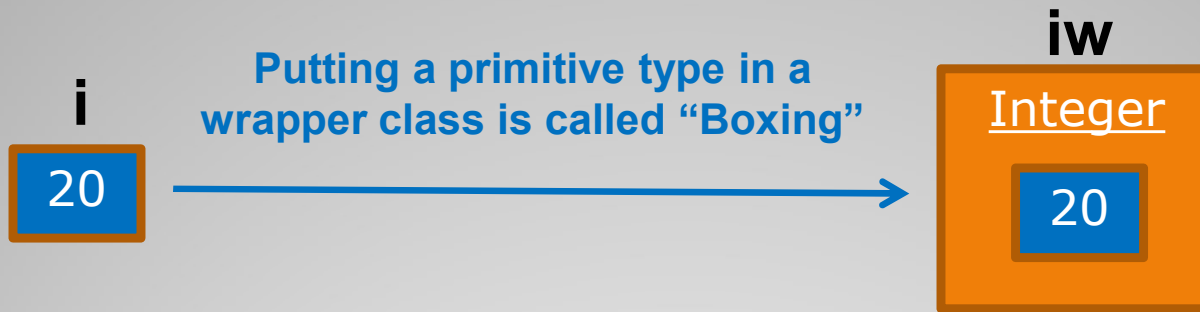
To mail a phone you must put it in a box (or package of some sort). The post office will not mail it unless it is properly packaged.

**Now phone
can be mailed**

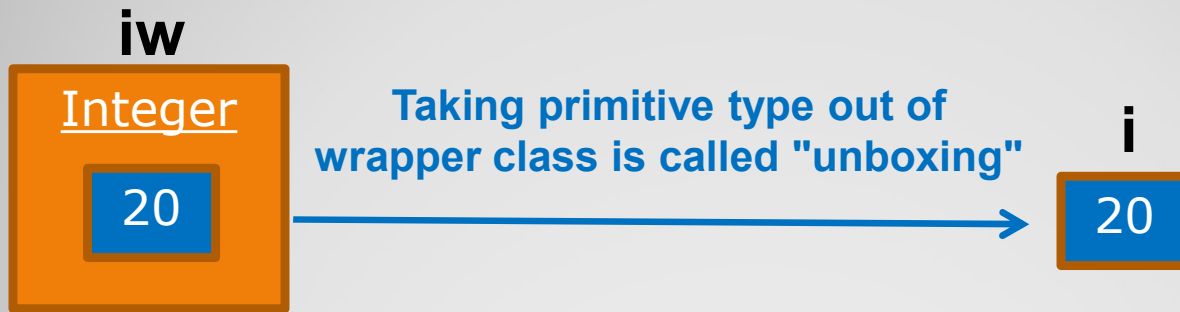


Wrapper Data Types

```
int i = 20; // Declare the primitive type
Integer iw; // Declare the wrapper instance
iw = new Integer(i); // Create instance and pass in primitive value
```



```
int value;
value = iw.intValue(); // Get primitive value from wrapper class
```



Wrapper Data Types

- An inner class is a class defined inside of another class.
- The inner class can be used as a "helper" for the outer class.
- For example...

Inner Classes

```
public class Car  
{
```

```
    public class Helper  
    {
```

```
        // Class Helper members here...
```

```
    }
```

```
    // Class Car members here...
```

```
}
```

```
public class Main {
```

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

```
        Car c;
```

```
        c = new Car();
```

```
        Car.Helper h;
```

```
        h = c.new Helper();
```

```
    }
```

Helper is an inner class. It is defined within Car (it does not have to be named Helper).

Create an instance of the outer class first

Create an instance of the inner class using the outer class instance. Call new with respect to the outer class instance.

Inner Classes

- Take Attendance!

Attendance