Java Programming

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

Today’s Lecture

- Chapter 6 (continued)
- Review methods
- Review stack and heap memory
- Call stack and activation records
- Method signatures
- Overloading

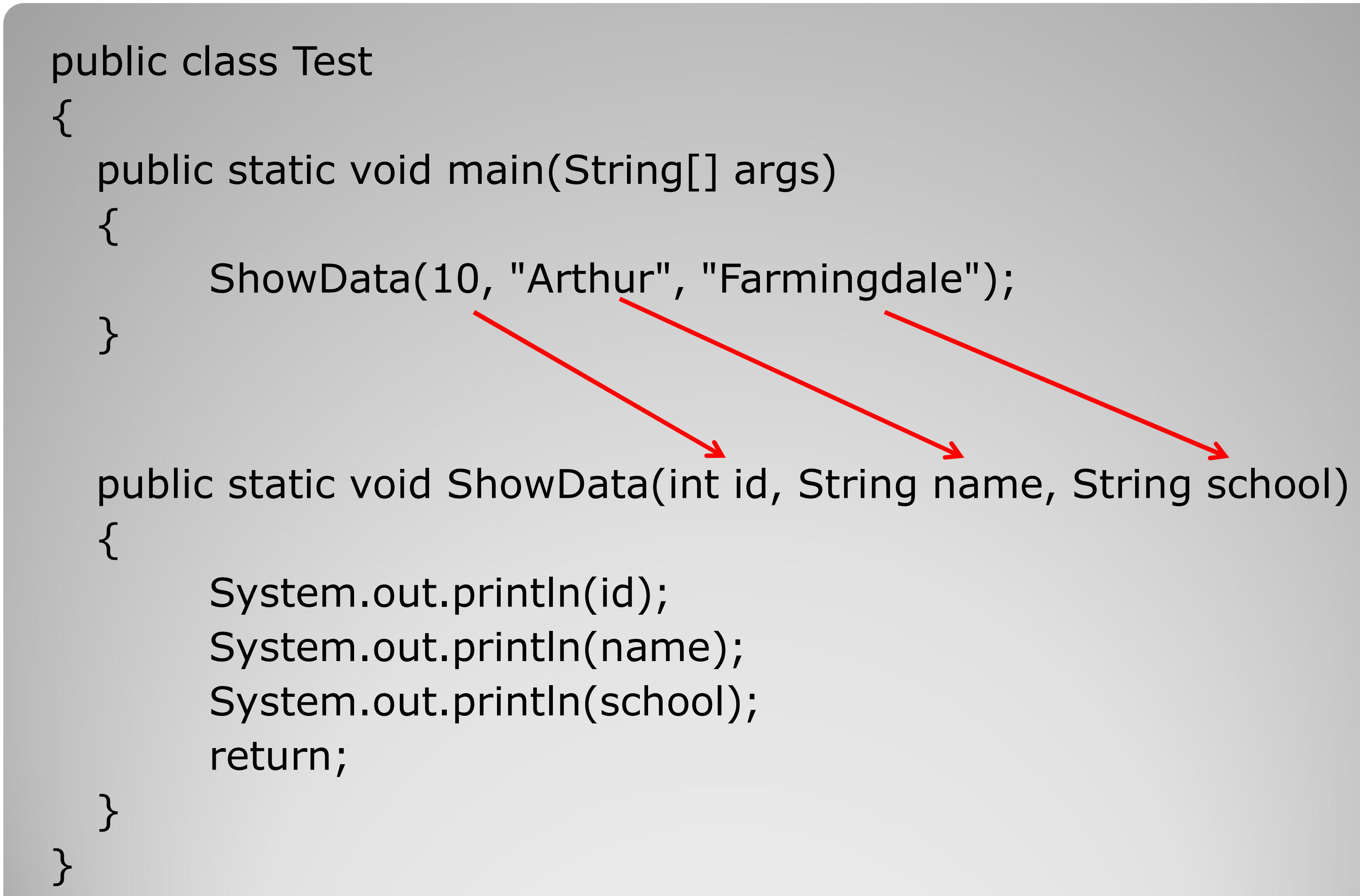
Created by Arthur Hoskey, PhD

- Method Review

Next Section

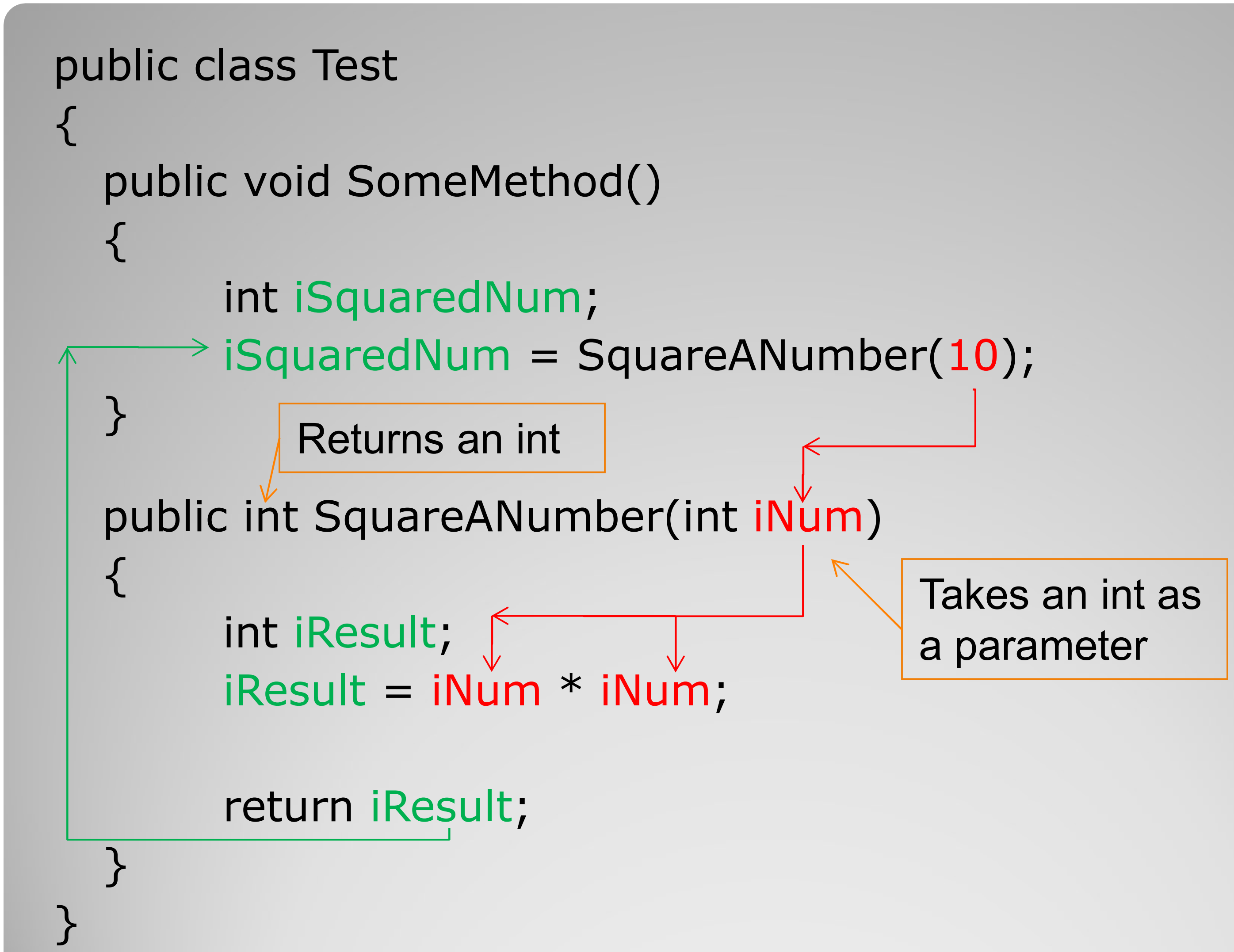
Created by Arthur Hoskey, PhD

Method With Multiple Parameters



```
public class Test
{
   public static void main(String[] args)
   {
          ShowData(10, "Arthur", "Farmingdale");
   }


   public static void ShowData(int id, String name, String school)
   {
          System.out.println(id);
          System.out.println(name);
          System.out.println(school);
          return;
   }
}
```

Methods and Assignment REVIEW



```
public class Test
{
   public void SomeMethod()
   {
          int iSquaredNum;
          iSquaredNum = SquareANumber(10);
   }

   public int SquareANumber(int iNum)
   {
          int iResult;
          iResult = iNum * iNum;

          return iResult;
   }
}
```

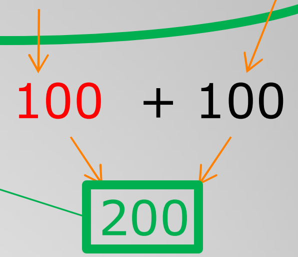
Returns an int

Takes an int as a parameter

Created by Arthur Hoskey, PhD

Methods and Assignment REVIEW

```
public class Test
{
   public void SomeMethod()
   {
         int iSquarePlusOneHundred;
         iSquarePlusOneHundred = SquareANumber(10) + 100;
   }

   public int SquareANumber(int iNum)
   {
         int iResult;
         iResult = iNum * iNum;

         return iResult;
   }
}
```



SquareANumber() evaluates to 100 which is then added to the constant 100 creating the value 200.

Created by Arthur Hoskey, PhD

- Stack and heap memory

Next Section

Created by Arthur Hoskey, PhD

Two types of Memory

Stack

All local variables and parameters

Heap

Member variables of reference types

- Memory layout example…
- Both primitive and reference types are included.

Memory

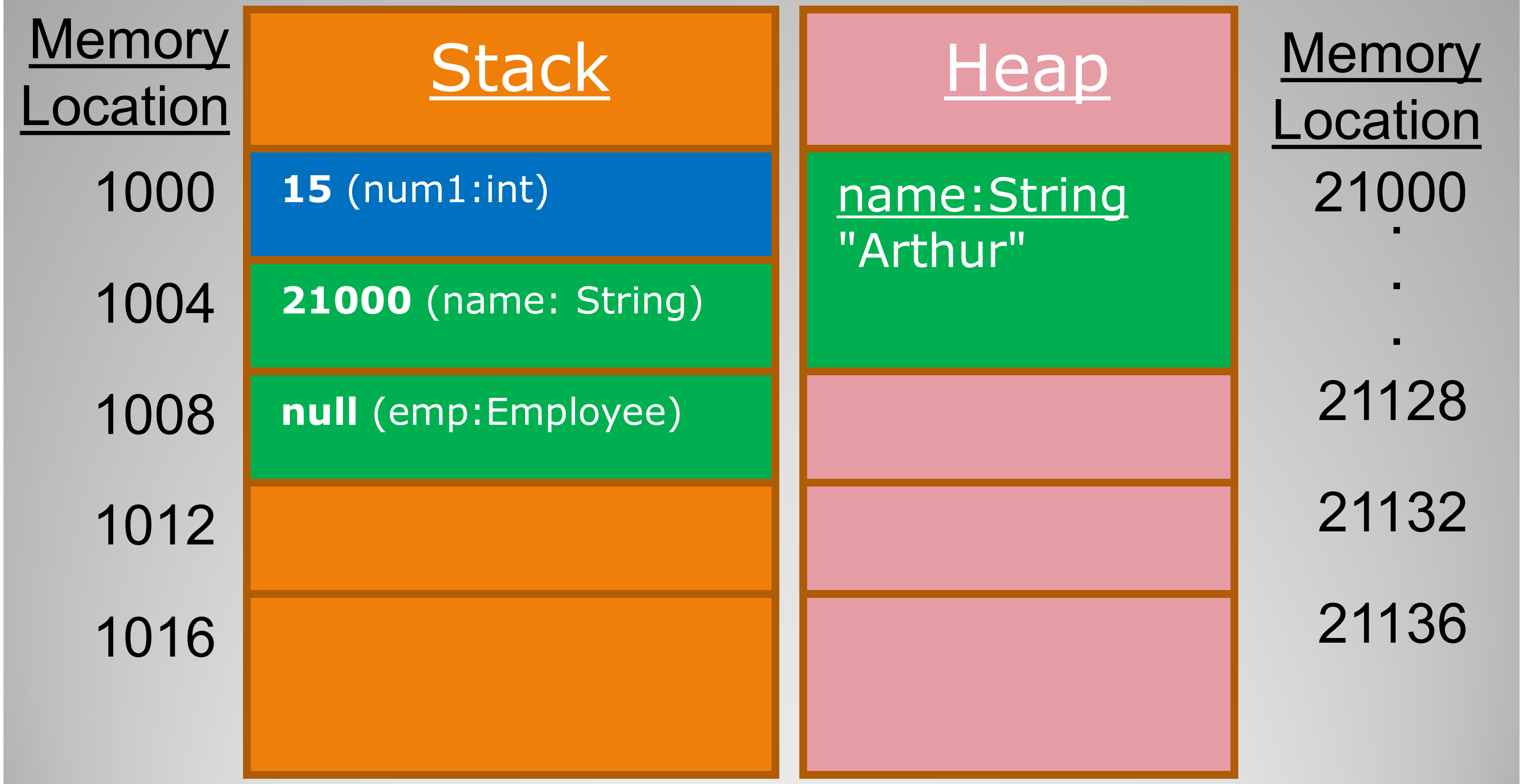
Memory

```
public class Employee {
   int   m_iId;
   int   m_iSalary;

   public Employee(int id, int salary) {
          m_iId = id;
          m_iSalary = salary;
   }

   public static void main(String[] args) {
          int num1 = 15;
          String name = new String("Arthur");
          Employee emp;
   }
};
```

What does memory look like?

Created by Arthur Hoskey, PhD

- ***Did not call new on Employee.***



Memory Location	Stack	Heap	Memory Location
1000	**15** (num1:int)	name:String "Arthur"	21000 . . .
1004	**21000** (name: String)		
1008	**null** (emp:Employee)		21128
1012			21132
1016			21136

Memory

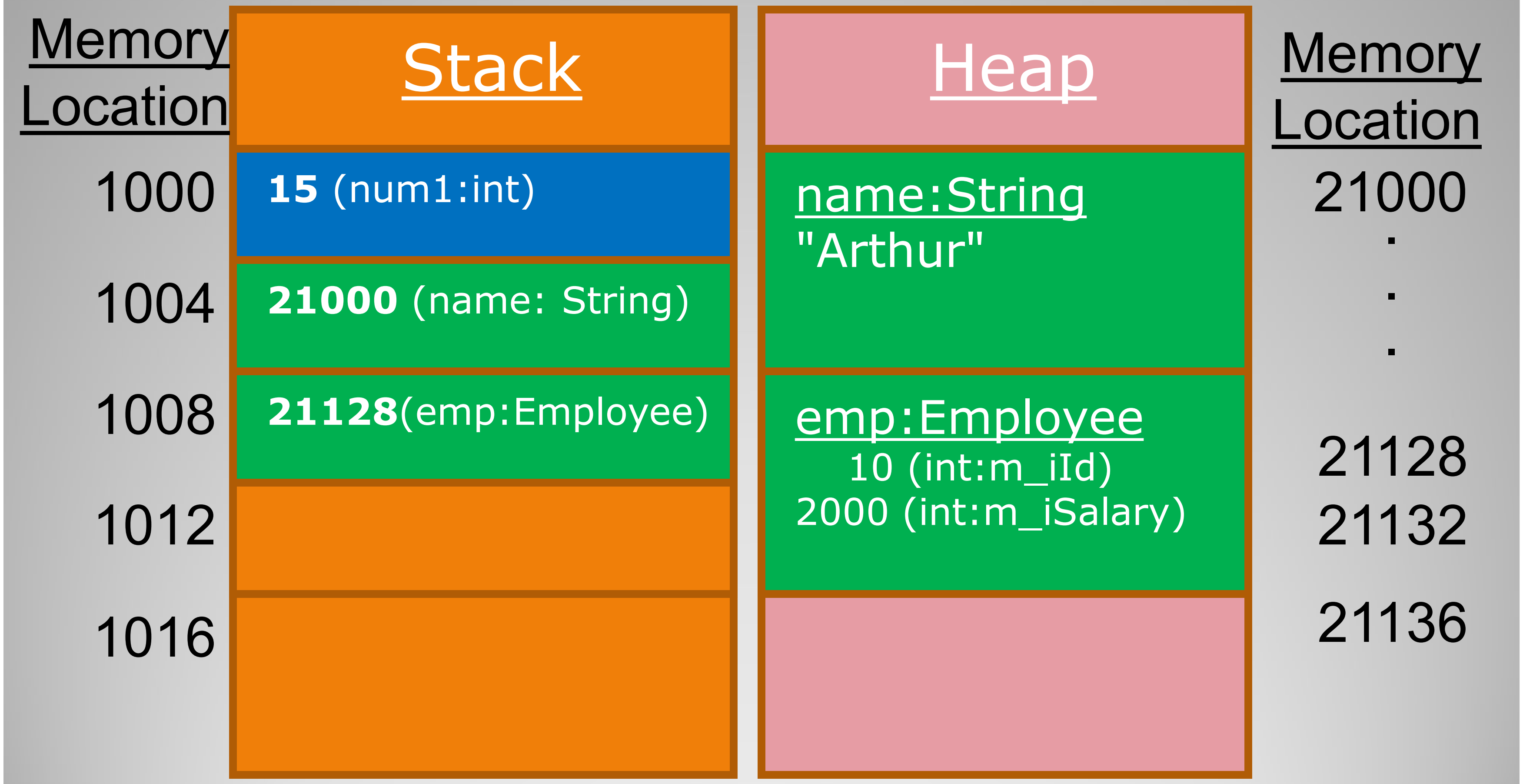
Created by Arthur Hoskey, PhD

Memory

```
public class Employee {
   int m_iId;
   int   m_iSalary;

   public Employee(int id, int salary) {
         m_iId = id;
         m_iSalary = salary;
   }

   public static void main(String[] args) {
         int num1 = 15;
         String name = new String("Arthur");
         Employee emp = new Employee(10, 2000);
   }
};
```

What does memory look like?

- ***new is called Employee.***



Memory Location	Stack	Heap	Memory Location
1000	**15** (num1:int)	name:String "Arthur"	21000 . . .
1004	**21000** (name: String)		
1008	**21128**(emp:Employee)	emp:Employee 10 (int:m_iId) 2000 (int:m_iSalary)	21128
1012			21132
1016			21136

Memory

Created by Arthur Hoskey, PhD

Show the memory layout of the following:

```
public class Student {
   private int id = 1;
   private int credits = 12;
   public static void main(String args[]) {
         Student s = new Student();
         int num = 10;
         Student s2 = new Student();
}
```

Hints:

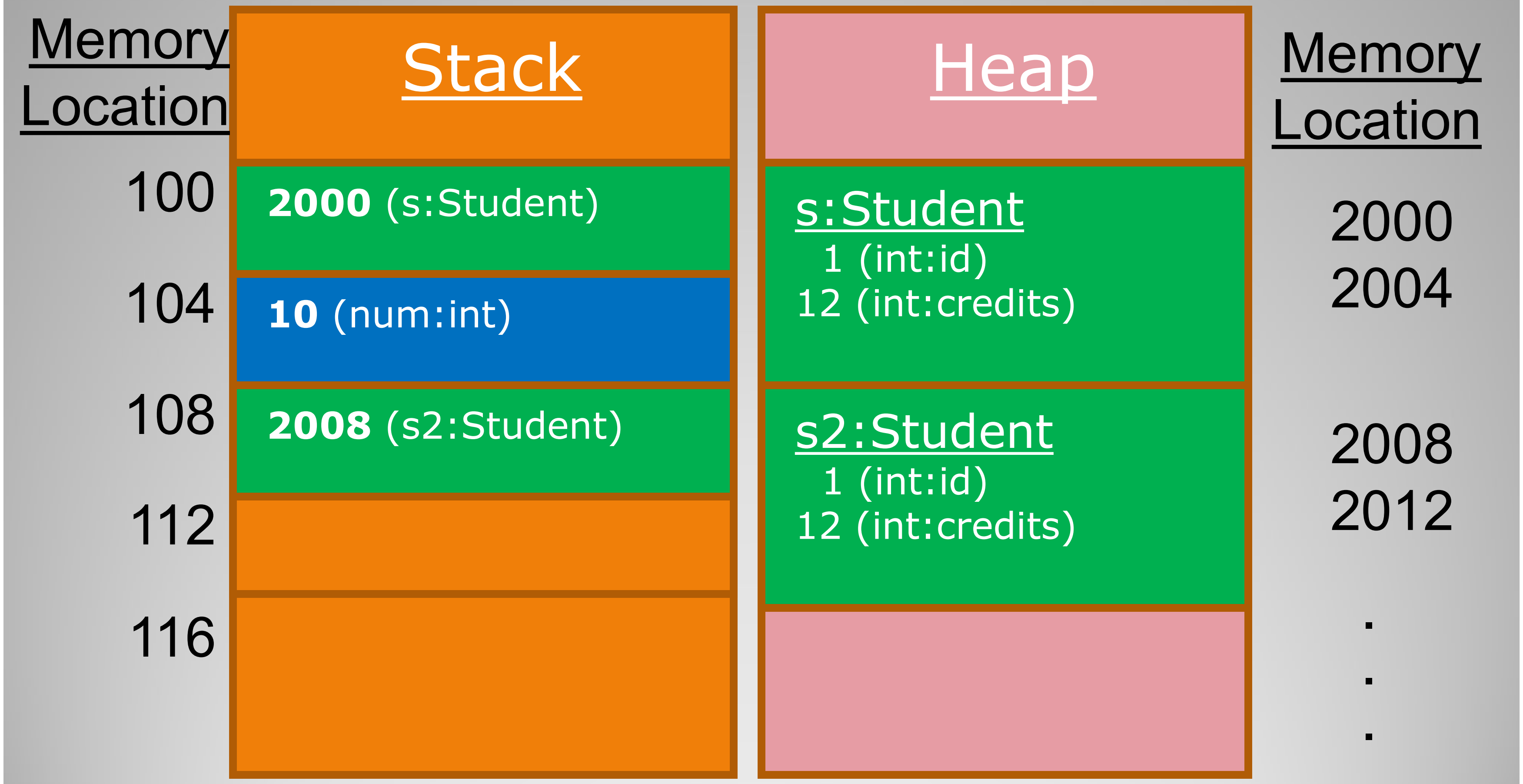
1. ***int variable takes up 4 bytes***
2. ***Reference pointer takes up 4 bytes***

	Stack		Heap
100	value (name, type)	2000	value (name, type)
104		2004	
108		2008	
112		2012	

Problem #1

Created by Arthur Hoskey, PhD

- ***SOLUTION***



Memory Location

Stack

100 **2000** (s:Student)

104 **10** (num:int)

108 **2008** (s2:Student)

112

116

Heap

Memory Location

s:Student

2000 1 (int:id)

2004 12 (int:credits)

s2:Student

2008 1 (int:id)

2012 12 (int:credits)

.
.
.

Memory

- Call stack and activation records

Next Section

Stacks

- A stack is a data structure (a collection of related items).

- Similar to a "stack of dishes".

- If you add a dish to the pile it will always be placed on top.

Created by Arthur Hoskey, PhD

Assume the following:

1. Only add to the top of the stack.

2. Only remove from the top of the stack.

- So, if you add a dish on top of a stack then that dish will be the first one removed (because it is on top).
- Last In First Out (LIFO). The last one in is the first one out.

Stacks

Stacks

- Terminology:
  - **Push**: Put something on the stack.
  - **Pop**: Take something off the stack.

- You ***push*** items on to a stack.
- You ***pop*** items off of a stack.

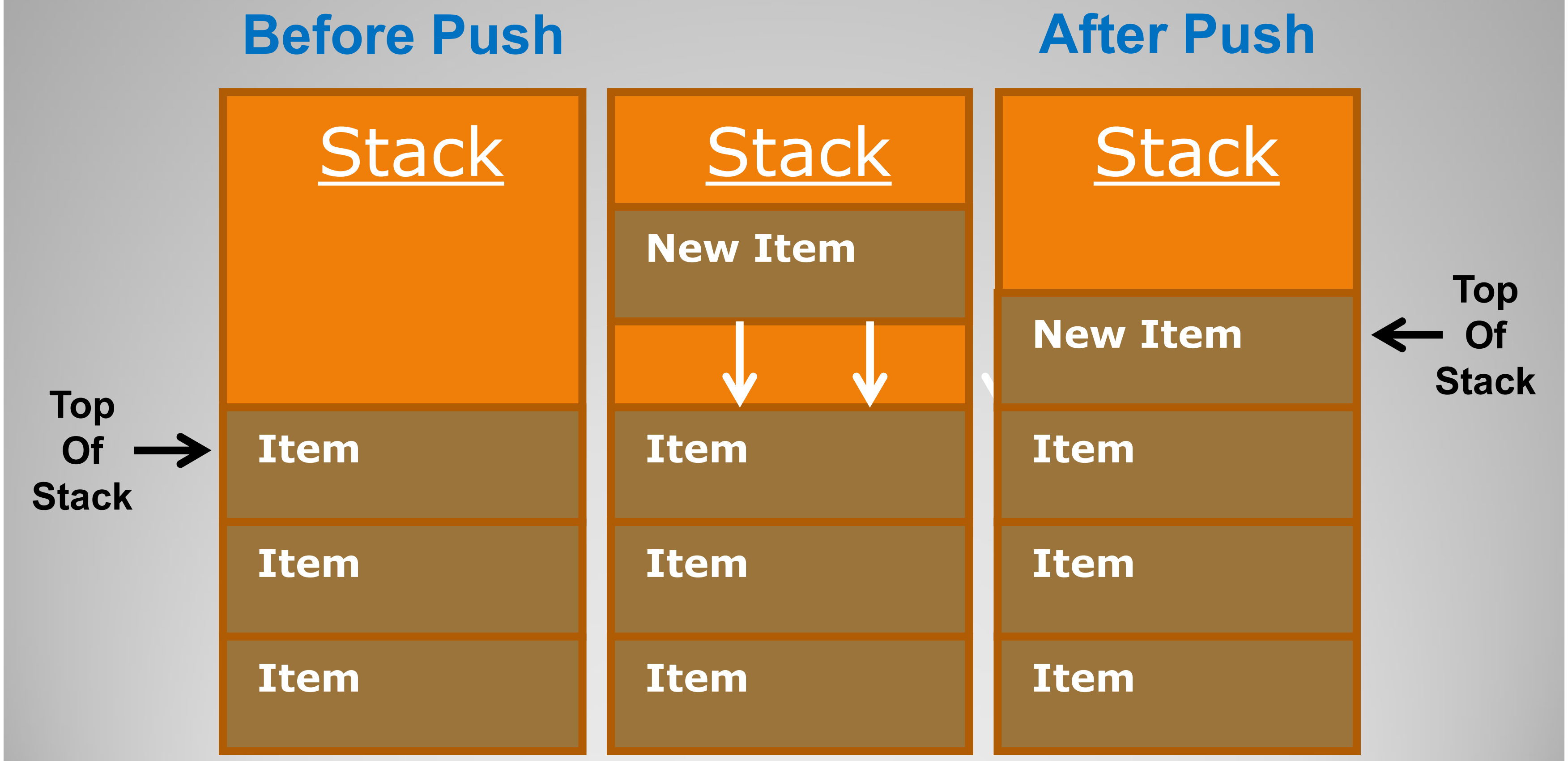
- Pushing and popping only occur from the top of the stack.

- For example…

Created by Arthur Hoskey, PhD

Stack - Push

- Add items – "Push" on to top of stack



Before Push

After Push

Stack

Top Of Stack

Item

Item

Item

Stack

New Item

Item

Item

Item

Stack

New Item

Top Of Stack

Item

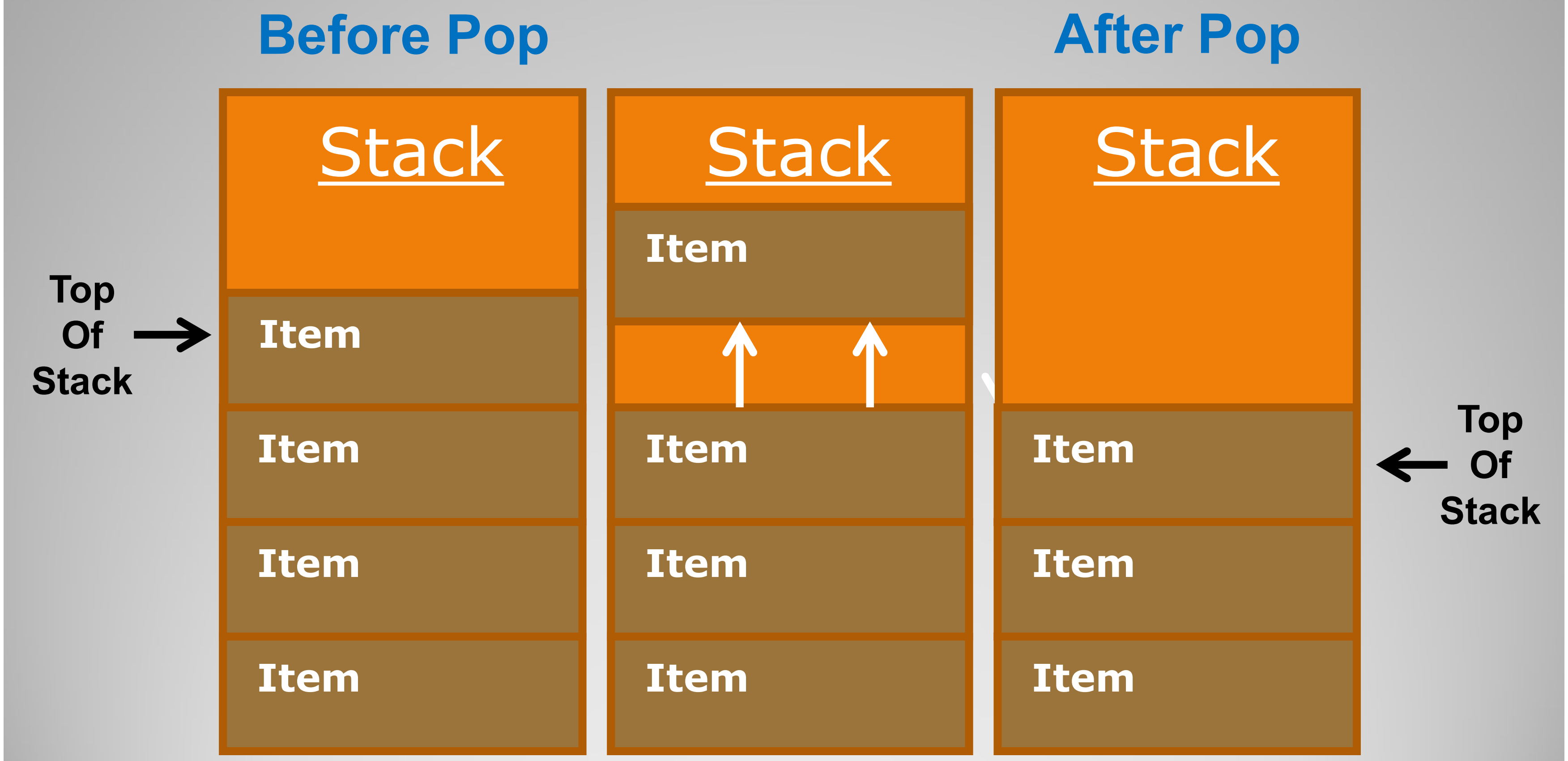
Item

Item

Created by Arthur Hoskey, PhD

Stack - Pop

- Remove items – "Pop" from top of stack



Before Pop

After Pop

Top Of Stack

Stack

Item

Item

Item

Item

Stack

Item

Item

Item

Item

Stack

Item

Item

Item

Top Of Stack

More details about the JVM stack.

- Proper name: ***Method call stack*** or ***program execution stack***.
- Variables are not just stored anywhere on the stack.
- Variables from the ***same method*** are grouped together on the stack.

Method Call Stack

Created by Arthur Hoskey, PhD

Method Call Stack

- All variables declared in a method are stored in an ***activation record (or stack frame)***.
- The activation record for a method call stores all the variables declared in that method.
- Call Stack Actions
  - **Call Method: Push activation record on stack.**
  - **End Method: Pop activation record off stack.**
- For example…

Program has not started yet. No activation records on stack.

```
void B() {
   System.out.println("In B");
}

void A() {
   System.out.println("In A");
   B();
   B();
}

void main(…) {
 System.out.println("In main");
 A();
}
```

Call Stack

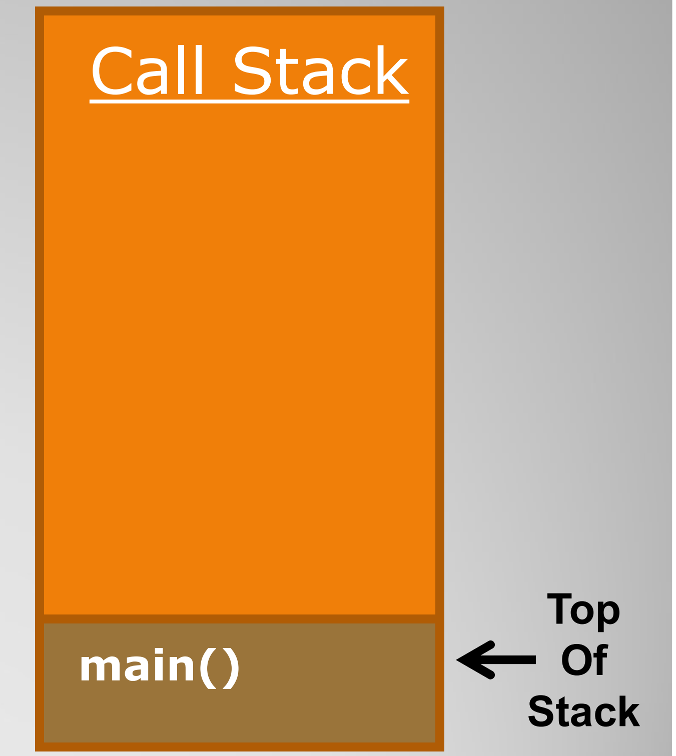
empty

Method Calls and Call Stack

Program started. In main and about to execute the "next" line (in bold).

```
void B() {
   System.out.println("In B");
}

void A() {
   System.out.println("In A");
   B();
   B();
}

void main(…) {
 System.out.println("In main"); // next
 A();
}
```

At "next"



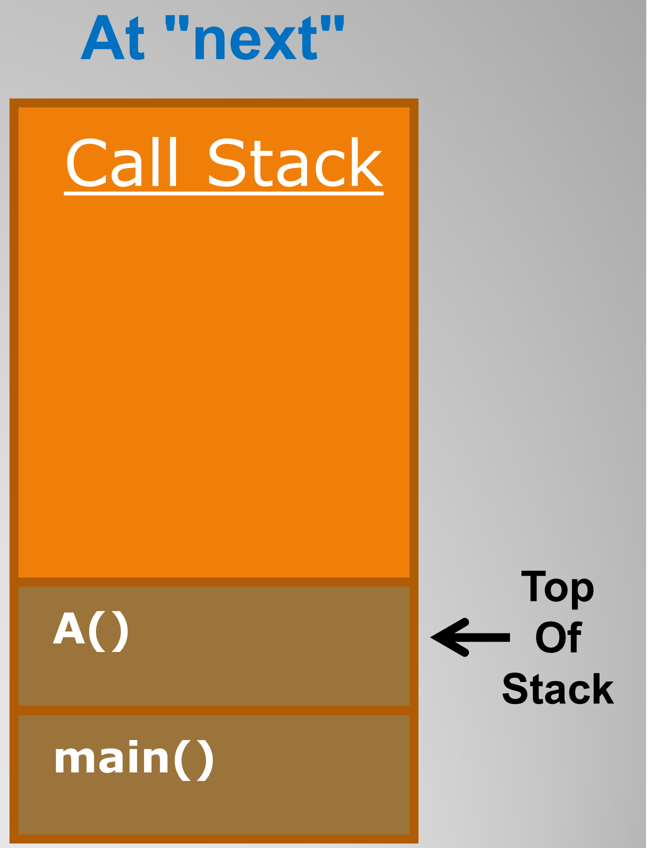
Method Calls and Call Stack

Created by Arthur Hoskey, PhD

Method Calls and Call Stack

Main called A. This causes an activation record for A to be pushed on stack.

```
void B() {
   System.out.println("In B");
}

void A() {
   System.out.println("In A");
   B(); // next
   B();
}

void main(...) {
 System.out.println("In main");
 A(); // called from here...
}
```



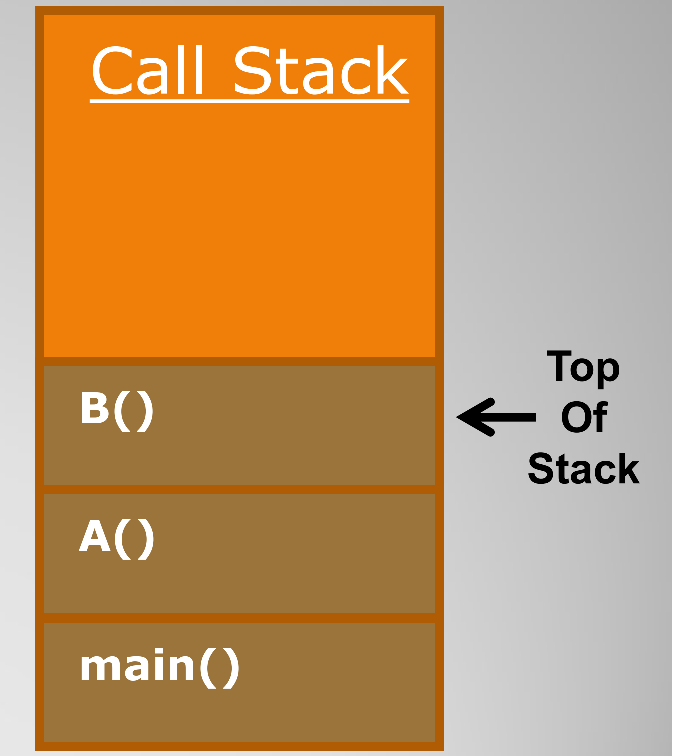
Created by Arthur Hoskey, PhD

Method Calls and Call Stack

A called B. This causes an activation record for B to be pushed on stack.

```
void B() {
   System.out.println("In B"); // next
}

void A() {
   System.out.println("In A");
   B(); // called from here…
   B();
}

void main(…) {
 System.out.println("In main");
 A();
}
```

At "next"



Call Stack

B()

A()

main()

Top Of Stack

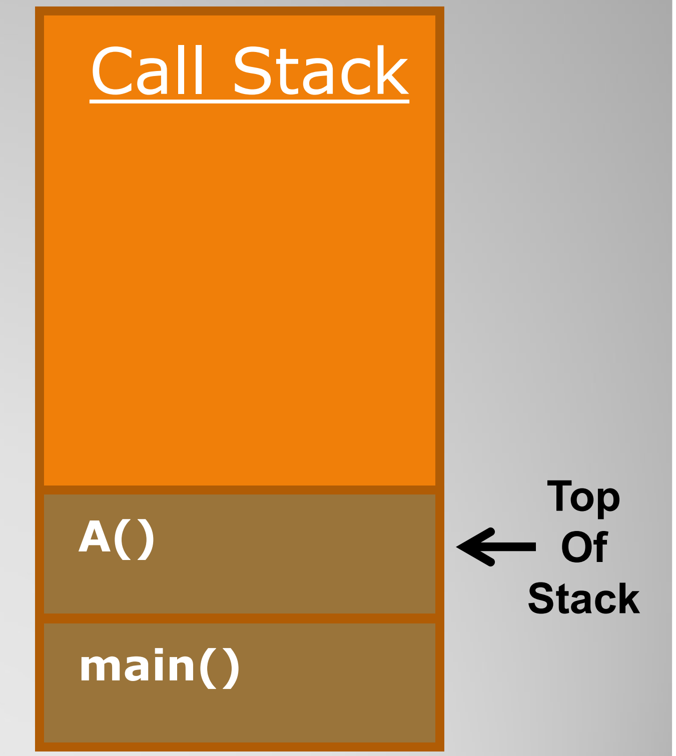
Created by Arthur Hoskey, PhD

Method Calls and Call Stack

B ended. This causes B activation record to be popped. A will call B again.

```
void B() {
   System.out.println("In B");
}

void A() {
   System.out.println("In A");
   B();
   B(); // next
}

void main(…) {
 System.out.println("In main");
 A();
}
```

At "next"

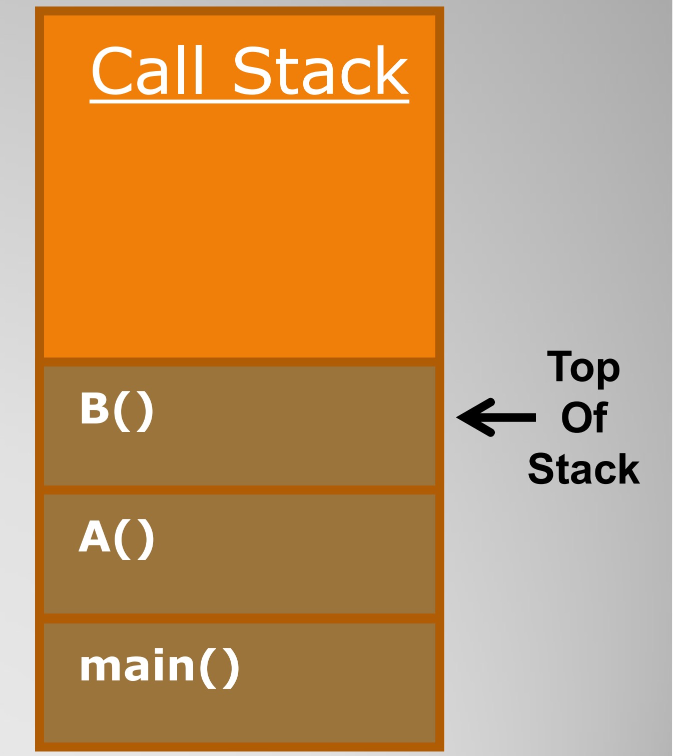


Created by Arthur Hoskey, PhD

A called B again. An activation record for B is pushed on the stack again.

```
void B() {
   System.out.println("In B"); // next
}

void A() {
   System.out.println("In A");
   B();
   B(); // called from here...
}

void main(...) {
 System.out.println("In main");
 A();
}
```

At "next"



Method Calls and Call Stack

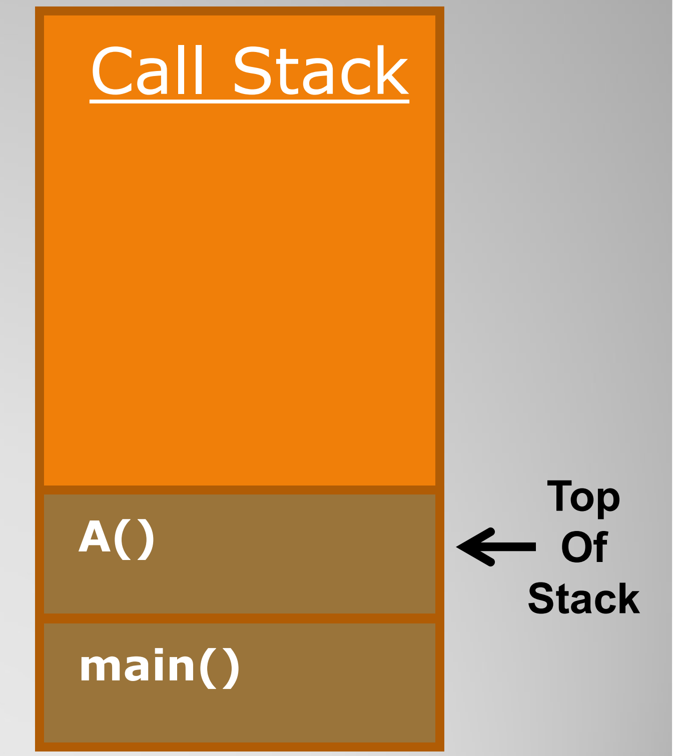
Created by Arthur Hoskey, PhD

Method Calls and Call Stack

B ended. This causes B activation record to be popped. A about to end.

```
void B() {
   System.out.println("In B");
}

void A() {
   System.out.println("In A");
   B();
   B();
} // next

void main(…) {
 System.out.println("In main");
 A();
}
```

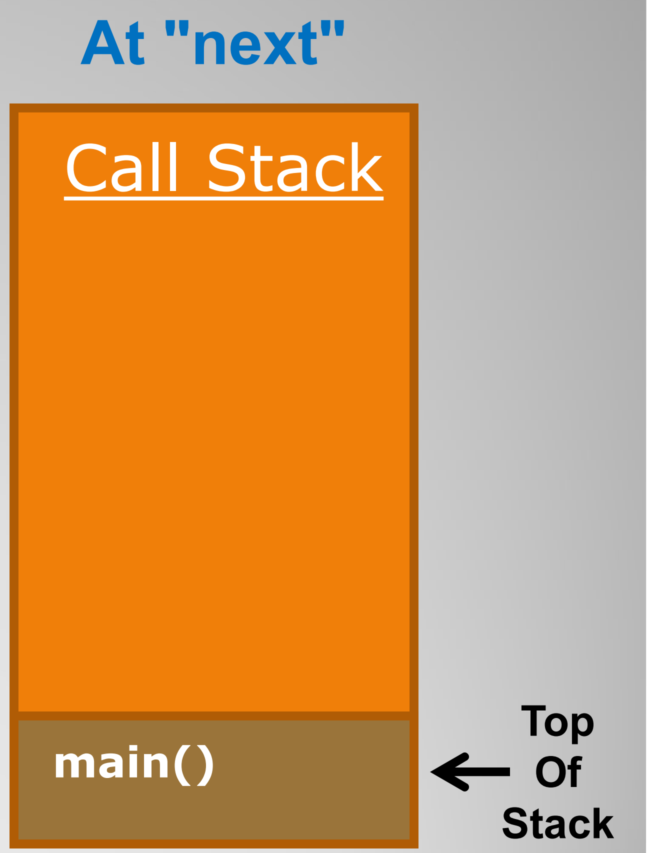
At "next"



Created by Arthur Hoskey, PhD

A ended. This causes A activation record to be popped. main about to end.

```
void B() {
   System.out.println("In B");
}

void A() {
   System.out.println("In A");
   B();
}

void main(…) {
 System.out.println("In main");
 A();
} // next
```



At "next"

Call Stack

main()

Top
Of
Stack

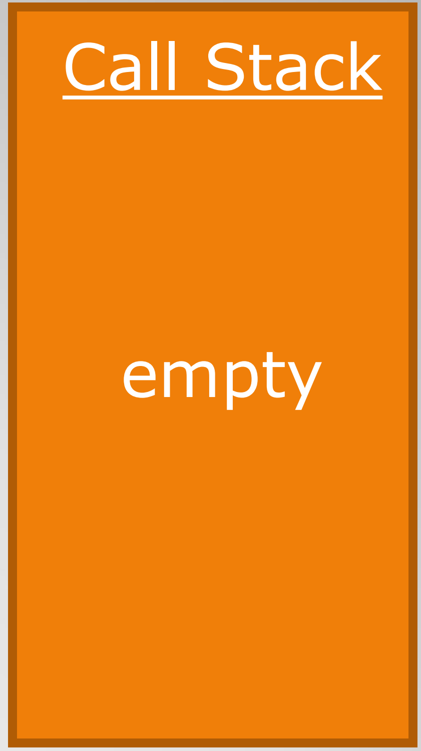
Method Calls and Call Stack

Created by Arthur Hoskey, PhD

main ended. Program Done. No more activation records on stack.

```
void B() {
   System.out.println("In B");
}

void A() {
   System.out.println("In A");
   B();
}

void main(…) {
 System.out.println("In main");
 A();
}
```

At "next"



Call Stack

empty

Method Calls and Call Stack

Created by Arthur Hoskey, PhD

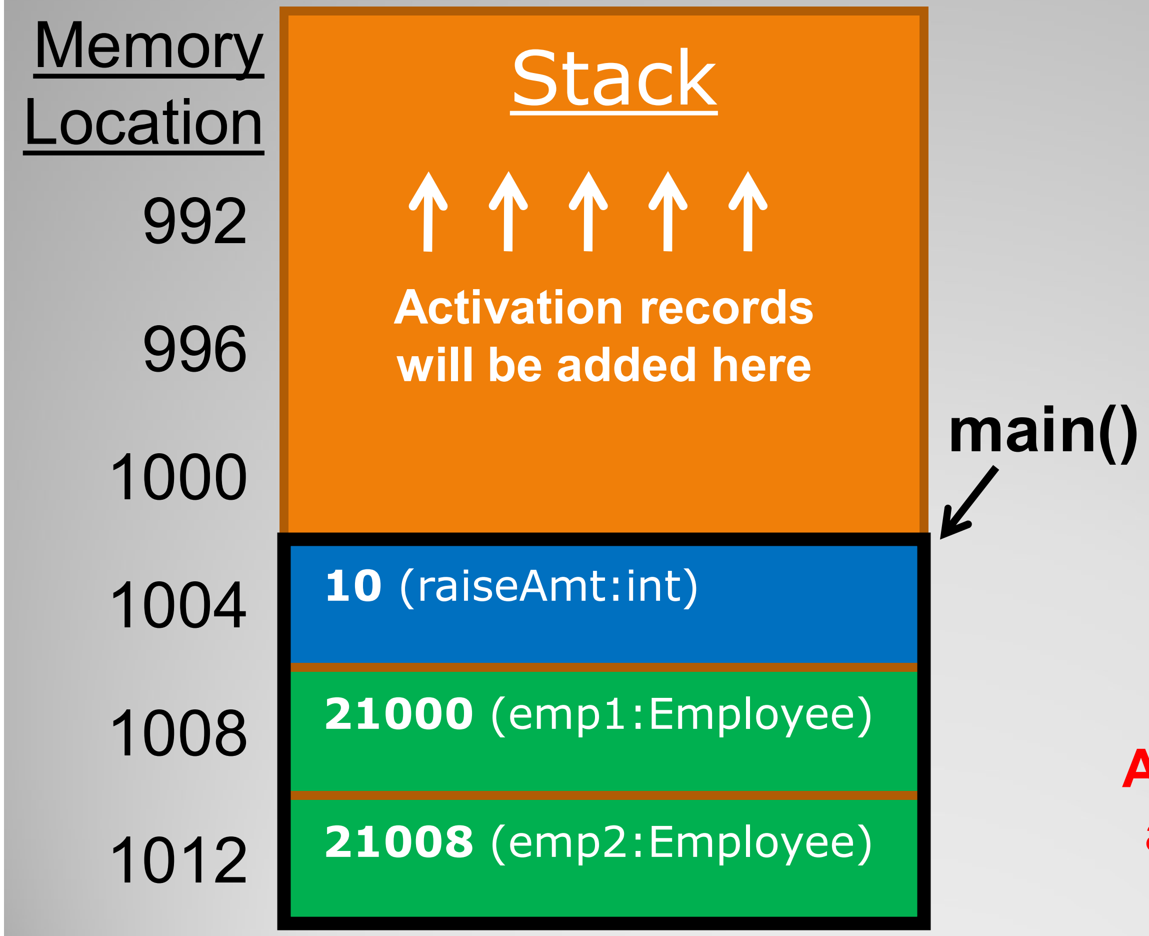
```
public class Employee {
    int m_Id;
    int m_Salary;

    public Employee(int id, int salary) {
            m_Id = id;
            m_Salary = salary;
    }

    public void Raise(int amount) {
            m_Salary = m_Salary + amount;
    }

    public static void main(…) {
            Employee emp1 = new Employee(111, 20);
            Employee emp2 = new Employee(222, 50);
            int raiseAmt = 10;
            emp1.Raise(raiseAmt); // next
            emp2.Raise(raiseAmt);
    }
};
```

Assume the program has executed to the "next" line.

What does memory look like in more detail using activation records?

Created by Arthur Hoskey, PhD

- **main local variables grouped together**



Memory Location	Stack
992	Activation records will be added here
996	
1000	
1004	**10** (raiseAmt:int)
1008	**21000** (emp1:Employee)
1012	**21008** (emp2:Employee)

main()

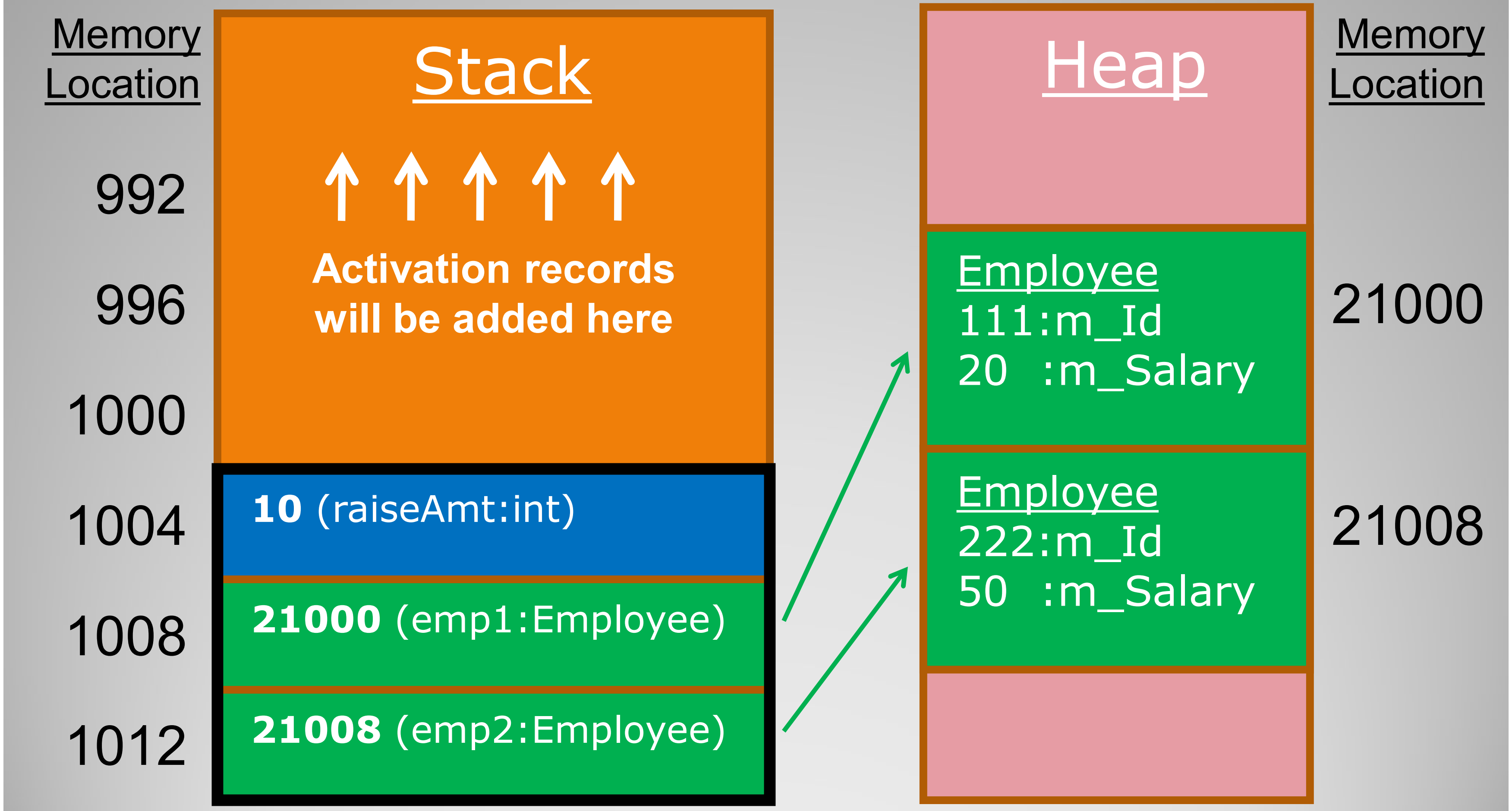
main() method's activation record

The activation record is colored black

Important
Activation record holds all local variables and parameters

Memory

- **emp1 and emp2 refer to heap locations**



Memory Location

992

996

1000

1004

1008

1012

Stack

Activation records will be added here

10 (raiseAmt:int)

21000 (emp1:Employee)

21008 (emp2:Employee)

Heap

Employee
111:m_Id
20 :m_Salary

Employee
222:m_Id
50 :m_Salary

Memory Location

21000

21008

Memory

Created by Arthur Hoskey, PhD

```
public class Employee {
    int m_Id;
    int m_Salary;

    public Employee(int id, int salary) {
            m_Id = id;
            m_Salary = salary;
    }

    public void Raise(int amount) {
            m_Salary = m_Salary + amount;
    }

    public static void main(…) {
            Employee emp1 = new Employee(111, 20);
            Employee emp2 = new Employee(222, 50);
            int raiseAmt = 10;
            emp1.Raise(raiseAmt);
            emp2.Raise(raiseAmt);
    }
};
```

When inside the Raise method how does it know which m_Salary to use?

Is the value 20 or 50?

Created by Arthur Hoskey, PhD

- How does it know which m_Salary to use?

Answer: It passes in the base address of the instance to work with when Raise is called.

- In general, when an instance method is called the instances reference is passed inside the **this** reference.

- It is a hidden parameter that gets passed into the method.

this Reference

this Reference

- The **this** reference is used to get access to the current instances member variables.

- **this** is automatically populated with the address of the current instance when an instance method is called.

- The value of **this** will change depending on which instance it was called from.

Created by Arthur Hoskey, PhD

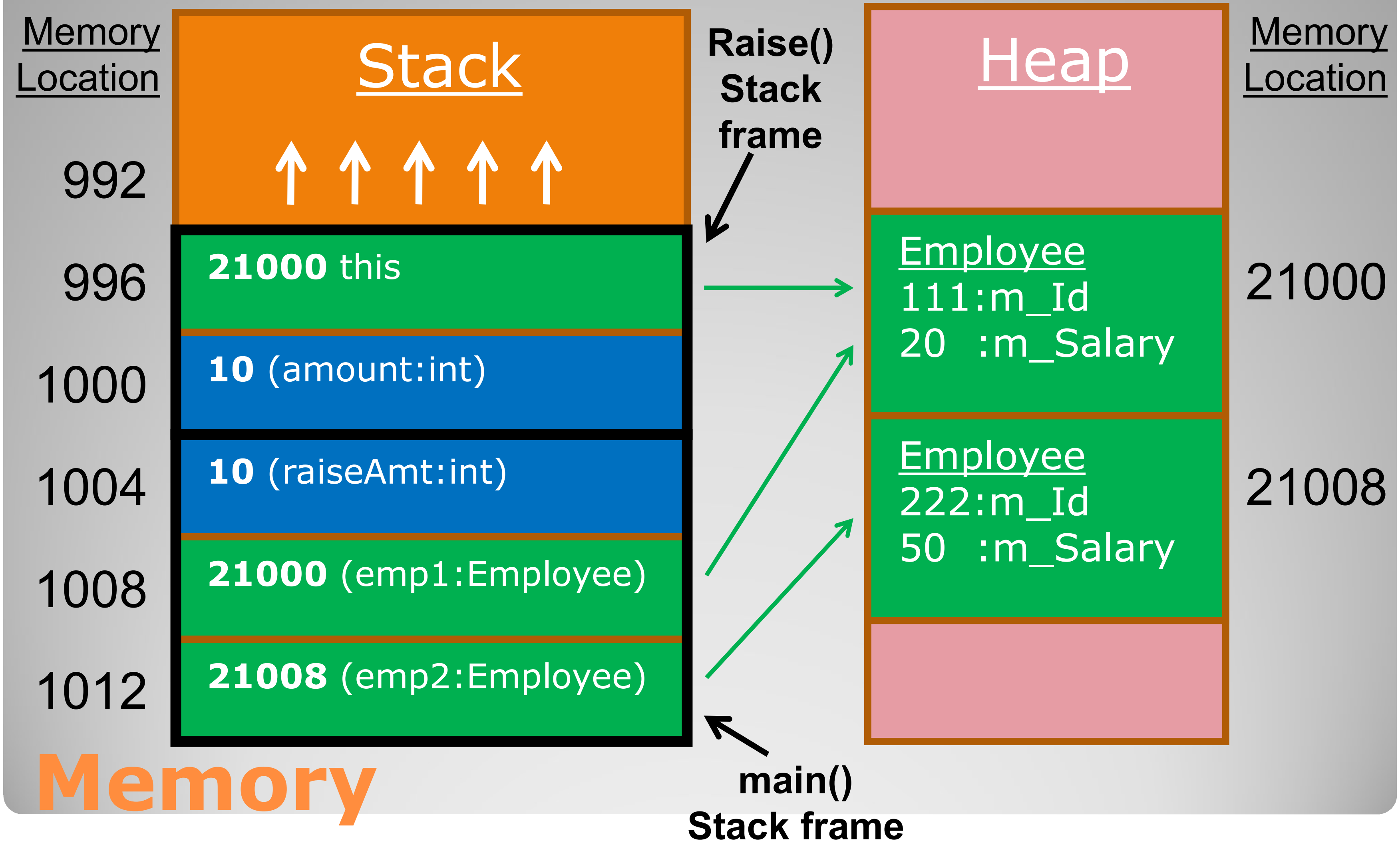
```
public class Employee {
   int m_Id;
   int m_Salary;

   public Employee(int id, int salary) {
          m_Id = id;
          m_Salary = salary;
   }

   public void Raise(int amount) {
          m_Salary = m_Salary + amount;
   }

   public static void main(…) {
          Employee emp1 = new Employee(111, 20);
          Employee emp2 = new Employee(222, 50);
          int raiseAmt = 10;
          emp1.Raise(raiseAmt); // called from here
          emp2.Raise(raiseAmt);
   }
};
```

When inside the Raise method how does it know which m_Salary to use?

Is the value 20 or 50?

Created by Arthur Hoskey, PhD

- **this reference has 21000 (m_Salary is 20)**



Memory Location

992

996

1000

1004

1008

1012

Stack

21000 this

10 (amount:int)

10 (raiseAmt:int)

21000 (emp1:Employee)

21008 (emp2:Employee)

Raise() Stack frame

main() Stack frame

Heap

Employee
111:m_Id
20 :m_Salary

Employee
222:m_Id
50 :m_Salary

Memory Location

21000

21008

Memory

Created by Arthur Hoskey, PhD

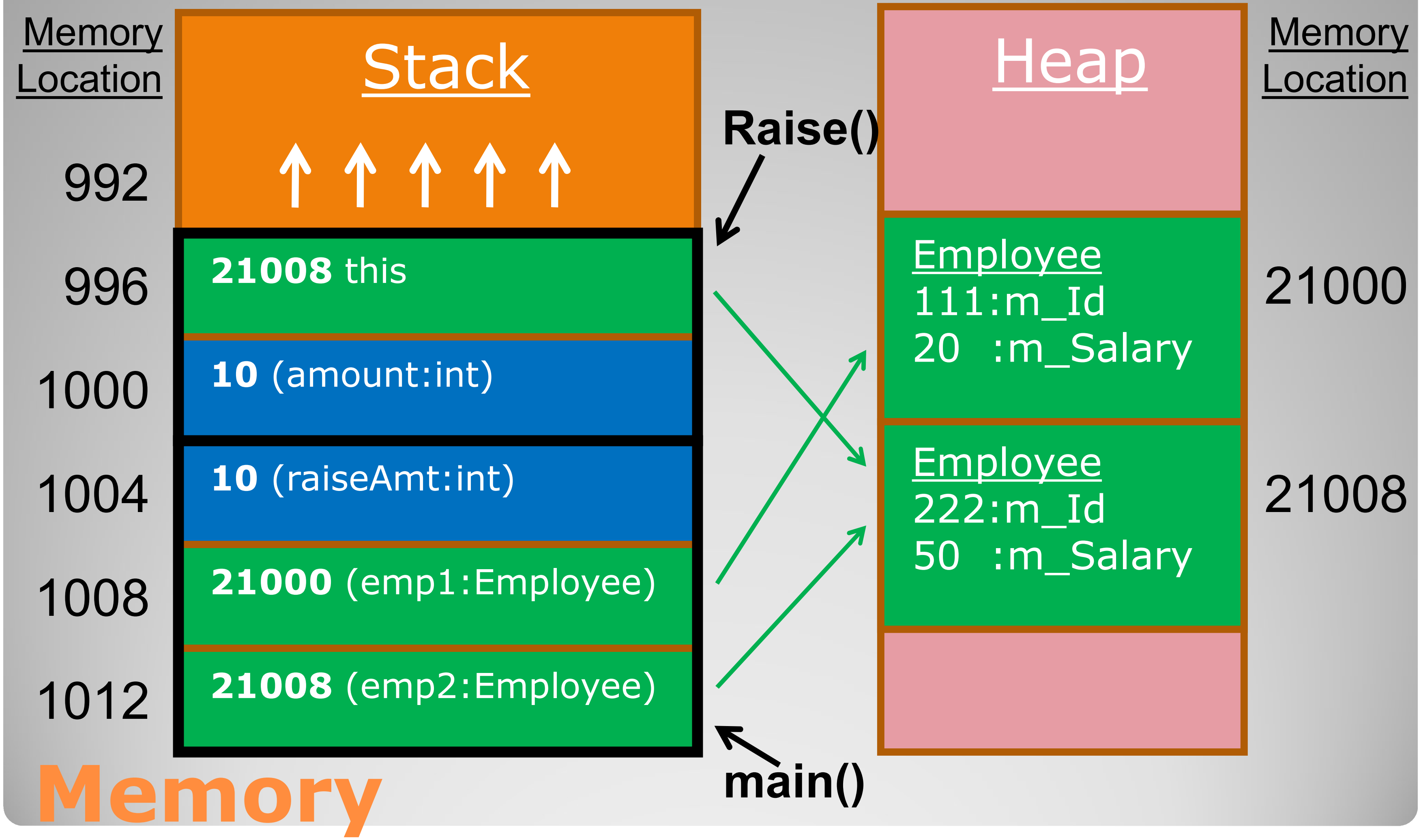
```
public class Employee {
    int m_Id;
    int m_Salary;

    public Employee(int id, int salary) {
            m_Id = id;
            m_Salary = salary;
    }

    public void Raise(int amount) {
            m_Salary = m_Salary + amount;
    }

    public static void main(…) {
            Employee emp1 = new Employee(111, 20);
            Employee emp2 = new Employee(222, 50);
            int raiseAmt = 10;
            emp1.Raise(raiseAmt);
            emp2.Raise(raiseAmt); // called from here
    }
};
```

When inside the Raise method how does it know which m_Salary to use?

Is the value 20 or 50?

Created by Arthur Hoskey, PhD

- **this reference has 21008 (m_Salary is 50)**



Memory Location

992

996

1000

1004

1008

1012

Stack

21008 this

10 (amount:int)

10 (raiseAmt:int)

21000 (emp1:Employee)

21008 (emp2:Employee)

Raise()

main()

Heap

Employee
111:m_Id
20 :m_Salary

Employee
222:m_Id
50 :m_Salary

Memory Location

21000

21008

Memory

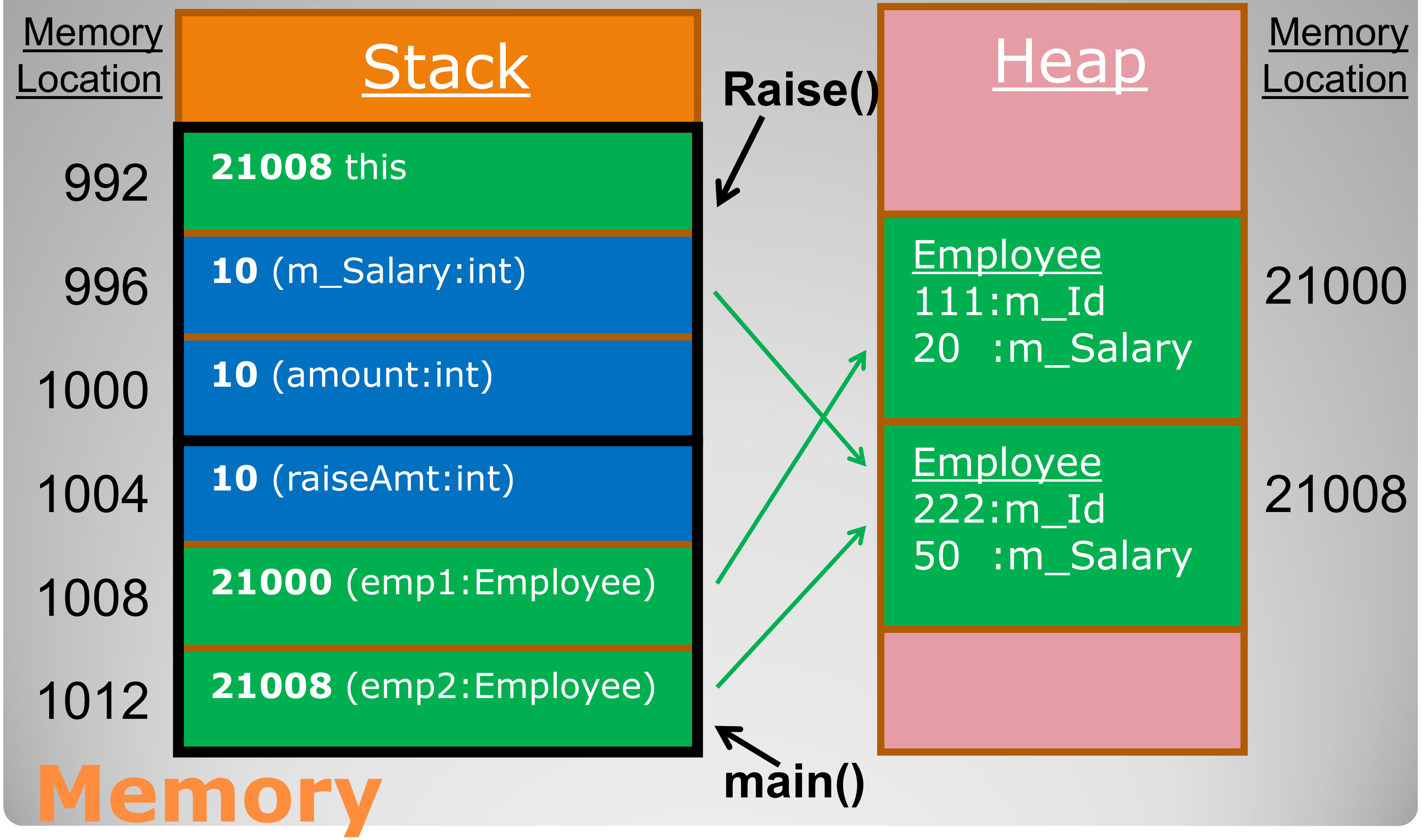
Created by Arthur Hoskey, PhD

```
public class Employee {
   int m_Id;
   int m_Salary;

   public Employee(int id, int salary) {
          m_Id = id;
          m_Salary = salary;
   }

   public void Raise(int amount) {
          int m_Salary;
          m_Salary = m_Salary + amount; // next
   }

   public static void main(…) {
          Employee emp1 = new Employee(111, 20);
          Employee emp2 = new Employee(222, 50);
          int raiseAmt = 10;
          emp1.Raise(raiseAmt);
          emp2.Raise(raiseAmt); // called from here
   }
};
```

Which m_Salary gets used?

What is the value of m_Salary before running "next" line?

- **Two m_Salary (local and member)**



Memory Location	Stack
992	**21008** this
996	**10** (m_Salary:int)
1000	**10** (amount:int)
1004	**10** (raiseAmt:int)
1008	**21000** (emp1:Employee)
1012	**21008** (emp2:Employee)

Heap	Memory Location
Employee 111:m_Id 20 :m_Salary	21000
Employee 222:m_Id 50 :m_Salary	21008

Memory

Created by Arthur Hoskey, PhD

Finding Correct Variable

Find the Correct Variable Inside a Method

1. Look for it as a local variable first (stored in activation record).

2. If not found then use **this** reference to find it as a member variable.

- If a variable is being used that is **not** declared in the current activation record it will follow the **this** reference and look for it as a member of the class.

Shadowing

- BE CAREFUL !!!
- The local variable m_Salary hides or "shadows" the member variable m_Salary.

```
public class Employee {
    int m_Id;
    int m_Salary;

    // other code here…

    public void Raise(int amount) {
            int m_Salary;   // Shadows member variable
            m_Salary = m_Salary + amount;
    }

    // other code here…
}
```

This will change the local m_Salary. The member variable m_Salary will remain unchanged.

Created by Arthur Hoskey, PhD

Shadowing

- You are allowed to explicitly use "this" in your code.
- Allows you to get around shadowing.

```
public class Employee {
    int m_Id;
    int m_Salary;

    // other code here…

    public void Raise(int amount) {
            int m_Salary;    // Shadows member variable
            this.m_Salary = this.m_Salary + amount;
    }

    // other code here…
}
```

You can explicitly use the "this" reference to avoid the shadowing

Created by Arthur Hoskey, PhD

- Do in-class problem for ch 6 p2.

In-Class Problem

- Method signatures
- Overloading

Next Section

Created by Arthur Hoskey, PhD

Method Signature

- Signatures identify methods.
- Method **signature** consists of two pieces:
  1. Method name
  2. Method parameters
- Method signatures must be unique within a given scope (for example inside a class).
- Cannot have two methods with the same signature ***in the same scope***.
- Return type is NOT part of the signature!

Created by Arthur Hoskey, PhD

- What are the method signatures?

```
public class Test
{
   public void H() { System.out.println("Hello"); }
   public void G() { System.out.println("Goodbye");
   public void I(int num) { System.out.println(num); }
   public void J(String s, int num) {
          System.out.printf("%s  %d\n", s, num);
   }
   public void K(int num, String s) {
          System.out.printf("%s  %d\n", s, num);
   }
}
```

- The method signatures are:

Signature	Name	Parameters
H()	H	none
G()	G	none
I(int num)	I	int
J(String s, int num)	J	String, int
K(int num, string s)	K	int, String

- **Is this legal? Are methods ambiguous?**

```
public class Test
{
  public void H()
  {
        System.out.println("Hello");
  }

  public void G() {
        System.out.println("Goodbye");
  }
}
```

- **YES. It is legal.**

```
public class Test
{
  public void H()
  {
       System.out.println("Hello");
  }

  public void G() {
       System.out.println("Goodbye");
  }
}
```

LEGAL. Same parameter lists *but* different names so OK.

- **Is this legal?**

```
public class Test
{
  public void H()
  {
       System.out.println("Hello");
  }

  public void H() {
       System.out.println("Goodbye");
  }
}
```

- **NO. It is not legal.**

```
public class Test
{
  public void H()
  {
        System.out.println("Hello");
  }

  public void H() {
        System.out.println("Goodbye");
  }
}
```

NOT LEGAL. Same parameter lists *and* same names.

Cannot distinguish between the two.

Created by Arthur Hoskey, PhD

- **Is this legal?**

```
public class Test
{
  public void H()
  {
        System.out.println("Hello");
  }

  public void H(String m) {
        System.out.println("Goodbye");
  }
}
```

- **YES. *It is legal*!!!**

```
public class Test
{
  public void H()
  {
       System.out.println("Hello");
  }

  public void H(String m) {
       System.out.println("Goodbye");
  }
}
```

LEGAL. Same name *but* different parameter list so OK.

Signatures are different!

Overloading

- **Overloading**
- Same name ***but*** different parameter lists.
- Two methods can have the same name in the same scope as long as they have different parameter lists.
- If the parameter lists differ then the signatures will differ even if the method name is the same.

Created by Arthur Hoskey, PhD

- Is this legal?

```
public class Test
{
  public void H()
  {
        System.out.println("Hello");
  }

  public int H() {
        System.out.println("Goodbye");
        return 10;
  }
}
```

Created by Arthur Hoskey, PhD

- **NO. *It is NOT legal*!!!**

```
public class Test
{
  public void H()
  {
      System.out.println("Hello");
  }

  public int H() {
      System.out.println("Goodbye");
      return 10;
  }
}
```

NOT LEGAL. Same name *and* same parameter list.

Return type is NOT part of the method signature!

Initialization - REVIEW

- ***How do we initialize a variable?***

- For **primitive** types it is easy:

```
int hourlyWorked = 35;

double hourlyRate = 35.50;

bool hourlyEmployee = true;
```

Initialization - REVIEW

- Reference types are tricker.
- A special method called a ***constructor*** is used to initialize an instance of an object.
- Constructors are called when you call new on the object being created.
- For example…

Constructor - REVIEW

```
public class Person {
  private int m_Age;

  public Person()
  {
       m_Age = 10;
  }
}

Person p;
p = new Person(); // Calls constructor
```

- Default constructor takes no parameters.
- You can also create constructors that take parameters.
- For example…

Constructor - REVIEW

Constructor - REVIEW

```
public class Person {
  private int m_Age;

  public Person(int age)
  {
       m_Age = age;
  }
}

Person p;
p = new Person(20); // Pass value into constructor
```

Created by Arthur Hoskey, PhD

Overloading Constructor

- The name of the constructor is the name of the class.
- Can you create more than one constructor for a class? **YES!!!**
- What must be different about each constructor?
- You can have as many constructors as you like as long as **ALL** the method signatures are unique.
- For example…

Overloading Constructor

```
public class Person {
   private int m_Age;

   public Person()  // Zero parameters
   {
          m_Age = 10;
   }

   public Person(int age)   // One parameter
   {
          m_Age = age;
   }
}
```

Attendance

Take Attendance!!!

Created by Arthur Hoskey, PhD