COMP-248
Object Oriented Programming I

Defining Classes II

By Emad Shihab, PhD, Fall 2015,
Parts of the slides are taken from Prof. L. Kosseim
Adapted for Section EE by S. Ghaderpanah, Fall 2015

Next:

1. **`static`** methods and variables
2. Wrapper classes
3. **References and class parameters**
4. Using and Misusing references
5. Packages and **`javadoc`**

References

- A variable can be
  - primitive type
  - reference to an object
- If n is an int, then n can contain a value of type int, e.g., 42
  - The value of the variable in stored in an assigned memory location
- If v is a variable of a class type:
  - then v does not contain an object of its class
  - v names the object and contains the memory address where the object is located

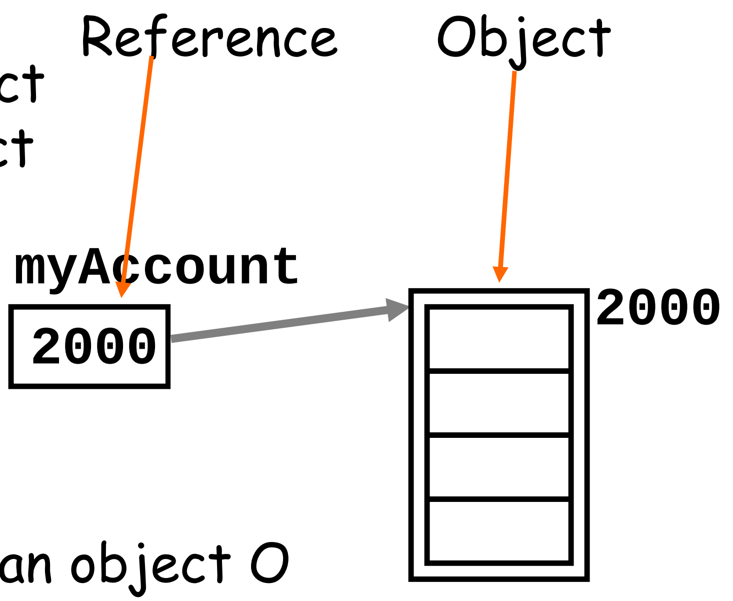
References

an object ≠ a reference to an object

reference=

holds the memory address of an object
can be seen as a "pointer" to an object

```
Account myAccount;
myAccount = new Account();
```



if a reference R contains the address of an object O
then we say that **R points to O**

References Example

```
public class ToyClass
{
    private String name;
    private int number;

    public ToyClass(String initialName, int
initialNumber)
    {
        name = initialName;
        number = initialNumber;
    }

    public ToyClass( )
    {
        name = "No name yet.";
        number = 0;
    }

     public String toString( )
    {
        return (name + " " + number);
    }

…
}
```

```
ToyClass var1 = new ToyClass(“Joe”, 42);
ToyClass var2;
var2 = var1;
var2.set(“Josephine”, 1);
System.out.println(var1.toString());
```

Driver

```
Josephine 1
```

Output

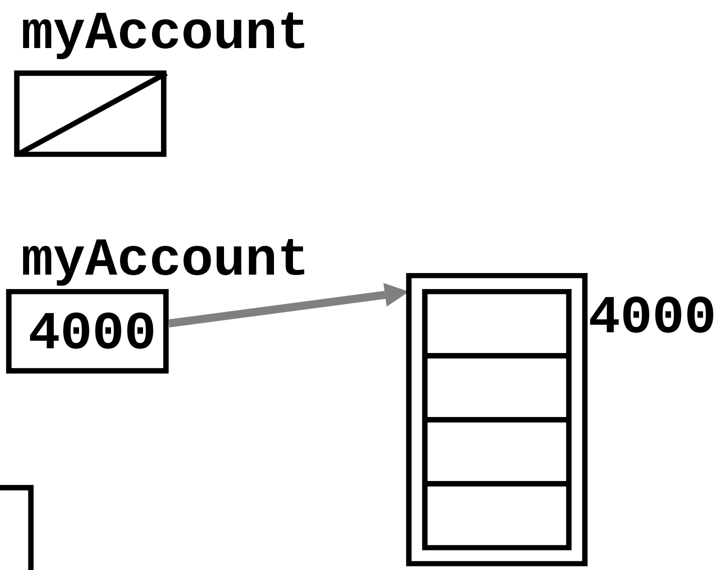
Two Special References

- `null` reference
- `this` reference

A null Reference

A reference that currently points to no object

```
Account myAccount;


myAccount = new Account();
```



```
String myName;

myName = new String("hello");
```

Accessing a null Reference

Be careful: a null reference points to no object so we cannot access its attributes or methods (cannot use the dot notation)

```
Account myAccount;
myAccount = new Account();
myAccount.deposit(100);  // 1. OK?
```

```
Account yourAccount;
yourAccount.deposit(100);  // 2. OK?
```

```
String myName;
System.out.print(myName.length());  // 3. OK?
```

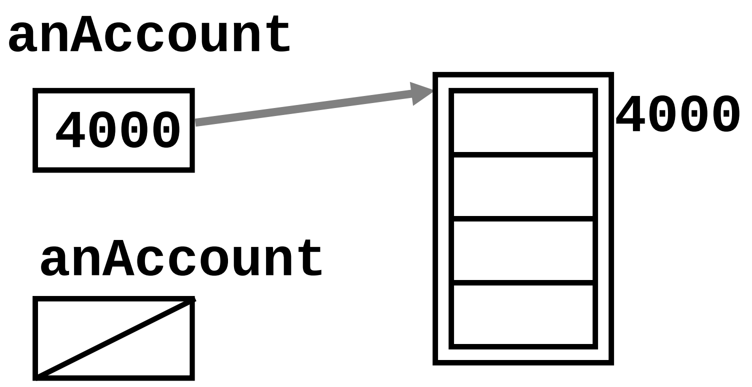
Checking for null References

- To avoid an error, we can check for null reference before accessing an object

```
if (anAccount!= null)
 anAccount.deposit(100);
```

- Can assign null to a reference

```
Account anAccount = new Account();
anAccount = null;
```



Just Checking ...

`null` can be used:

A. to indicate that a variable has no real value
B. in a Boolean expression with `==` or `!=`
C. as a placeholder
D. all of the above
E. none of the above

The this Reference

- The **this** reference
  - allows an object to refer to itself
  - is a reference to the current object

i.e. inside a method, **this** refers to the object through which the method is being executed

```
public void deposit(int amount)
{
   balance += amount;
  this.balance += amount;
}
```

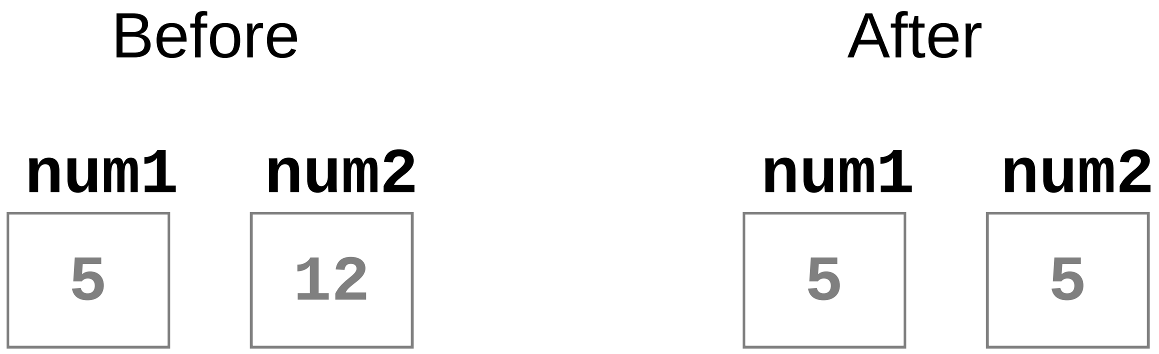
```
myAccount.deposit(100);   // "this" refers to ??
yourAccount.deposit(200); // "this" refers to ??
```

Assignment

The act of assignment takes a copy of a value and stores it in a variable

For primitive types:

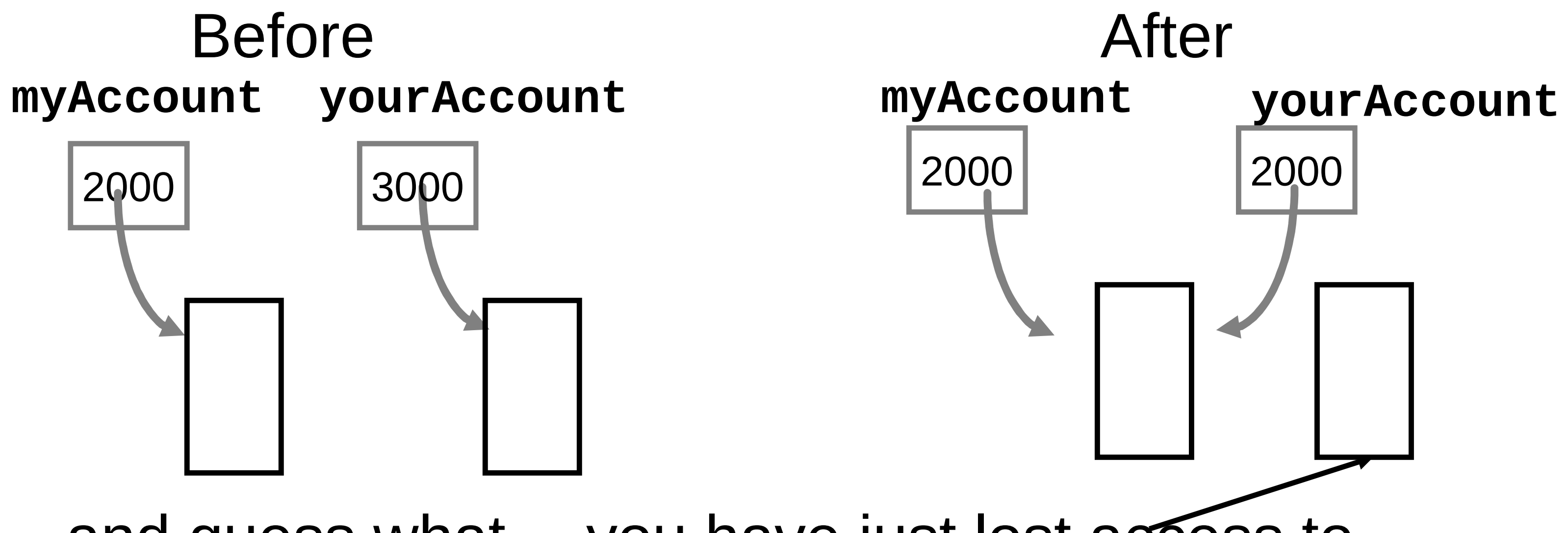
```
num2 = num1;
```



Reference Assignment

For <u>references</u>, assignment copies the memory location

```
Account myAccount = new Account();
Account yourAccount = new Account();
yourAccount = myAccount;
```

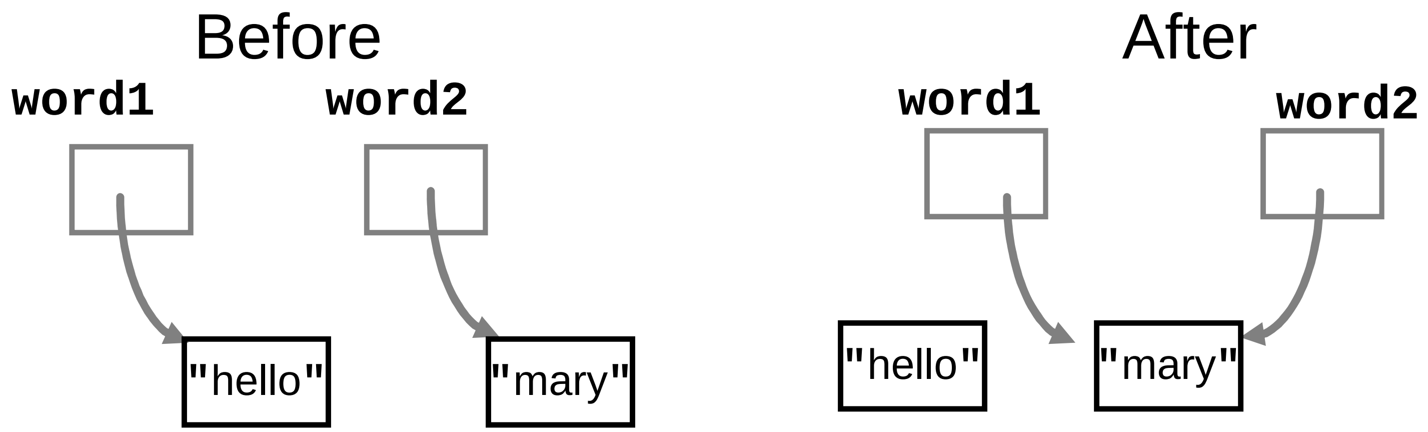


- and guess what… you have just lost access to
- if 2 references "point" to the same objects, they are aliases of each other

Reference Assignment

Same with strings… because strings are objects

```
String word1 = "hello";
String word2 = "mary";
word1 = word2;
```

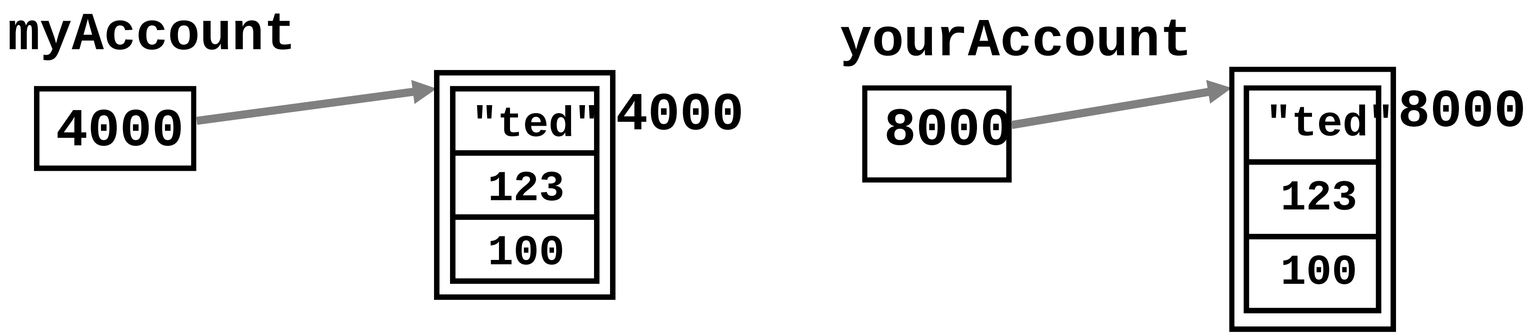


Equality of References

The == operator:

- compares equality of references
- returns **true** if the **references are aliases** of each other
  - ie if they **point** to the same object
  - **NOT if the objects pointed to have the same content**

```
Account myAccount = new Account("ted", 123, 100);
Account yourAccount = new Account("ted", 123, 100);
```



```
if (myAccount == yourAccount)    // true or false?
   System.out.print("the same");
```

Equality of Objects

The method **equals** :

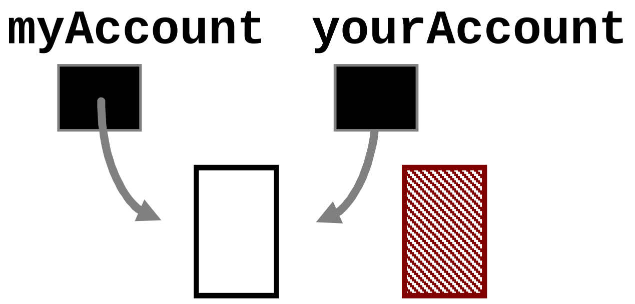
- is defined for all objects
- but unless we redefine it in our class, it has the same semantics as the == operator
- we can redefine it to return **true** under whatever conditions we think are appropriate (ex. equality of content, not address)

```
public class Account {
  private String name;
  private double balance;
  private int acctNumber;

  boolean equals(Account anotherAcc) {

     return (this.name.equals(anotherAcc.name) && (this.balance ==
     anotherAcc.balanace) && (this.acctNumber == anotherAcc.acctNumber))

  }
…
```

```
if (myAccount.equals(yourAccount))
   System.out.print("same content");
if (myAccount == yourAccount)
   System.out.print("same object");
```

Garbage Collection

- When an object no longer has any references to it, it can no longer be accessed by the program!



- Java has an automatic *garbage collector*
  - runs periodically
  - returns memory of inaccessible objects to the system for future use
- If a reference is no more useful… assign it to null (**`myRef=null;`**), so that the garbage collector can pick it up
- In other languages, the programmer is responsible for performing garbage collection

Pass by Value vs Pass by Reference

- Pass by value
  - all primitive types are always passed by value
  - the formal parameter is a copy of the actual parameter
  - the method modifies the copy
  - but the actual parameter is never modified
- Pass by reference
  - object parameters are always passed by reference
  - the actual parameter and the formal parameter become aliases of each other
  - the method can modify the actual parameters
  - we copy the reference; not the object

Example: Swapping 2 int

```
public class PassDriver
{
  public static void main(String[] arg)
  {
     int x = 10;
     int y = 20;

     System.out.println("1 " + x + " " + y);
     swap(x, y);
     System.out.println("4 " + x + " " + y);

   }

  public static void swap(int param1, int param2)
  {
    System.out.println("2 " + param1 + " " + param2);
    int temp = param1;
    param1 = param2;
    param2 = temp;
    System.out.println("3 " + param1 + " " + param2);
   }
}
```

```
1 10 20
2 10 20
3 20 10
4 10 20
```

Output

Example: Swapping 2 MyInt

PassDriver.java

```
public class PassDriver {
  public static void main(String[] arg) {
     MyInt a = new MyInt(10);
     MyInt b = new MyInt(20);
     System.out.println("1 " + a.getValue() + " " + b.getValue());
     swap(a, b);
     System.out.println("4 " + a.getValue() + " " + b.getValue());
   }

  public static void swap(MyInt param1, MyInt param2)
  {
    System.out.println("2 " + param1.getValue() + " " + param2.getValue());
    MyInt tmp = new MyInt(param1.getValue());
    param1.setValue(param2.getValue());
    param2.setValue(tmp.getValue());
    System.out.println("3 " + param1.getValue() + " " + param2.getValue());
   }
}
```

```
public class MyInt {
    private int value;

    public MyInt(int data) { this.value = data; }
    public void setValue(int data) { this.value = data; }
    public int getValue() { return value; }
}
```

MyInt.java

```
1 10 20
2 10 20
3 20 10
4 20 10
```

Output

Conclusion

if argument is a primitive type:

A method <u>cannot change</u> the value of the argument

if argument is a reference:

A method <u>can change</u> the value of an instance variable of an objet passed as argument

Anonymous Objects

- The **new** operator
  - calls a constructor which initializes an object,
  - returns a reference to the location in memory of the object created
- if the object created will only be used as an argument to a method, and never used again, no need to assign it to a variable
- create & use on the fly!

```
BankAccount temp = new BankAccount("mary", 100);
if (someObject.equals(temp)
   System.out.print("equal");

if (someObject.equals(new BankAccount("mary", 100))
    System.out.print("equal");
```

Next:

1. `static` methods and variables
2. Wrapper classes
3. References and class parameters
4. **Using and Misusing references**
5. Packages and `javadoc`

Example: Swapping 2 Account

```
public static void main(String[] arg)
{
  Account a = new Account("ted", 123, 100);
  Account b = new Account("mary", 456, 99);

  System.out.println(a + " " + b);
  swap(a, b);
  System.out.println(a + " " + b);
}

public static void swap(Account a, Account a)
{
      Account tmp;
      tmp = a;
      a = a;
      a = tmp;
}
```

Output