



CLASSES AND OBJECTS

©2015 Manipal Global Education



OBJECTIVES

*Classes and
Objects*



LEARNING OBJECTIVES

At the end of this lesson, you will be able to:

- Define classes and create objects
- Understand JVM Memory architecture
- Understand types of Variables
- Implement Overloading
- Use Access Modifiers





CLASSES AND OBJECTS - RECAP

➤ Class

- Template used to create objects
- Defines State and Behavior of an object
- State – Instance variables declared within a class
- Behavior –Instance methods or functions declared in a class
- A class encapsulates the data and functionality

➤ Objects

- instances created using a class
- Class is said to be instantiated when an object is created
- Each object has its own state stored in member or instance variables
- Exposes its behavior through methods
- Has unique identity



CONCEPT

Components of a class



COMPONENTS OF A CLASS

```
public class Account {  
    static int acctCount = 0;  
  
    int accountNo;  
    double accountBalance;  
    String accountType;  
  
    public Account(String acType) {  
        accountNo = ++ Account.acctCount;  
        accountBalance = 0;  
        accountType = acType;  
    }  
  
    public void depositAmount(double amount) {  
        accountBalance += amount;  
    }  
  
    public String getAccountDetails() {  
        String format = "Acct No : "+accountNo  
        +"\\n" + "Acct Balance: "+accountBalance;  
        return format;  
    }  
}
```

Static
Variable

Instance
Variables

Construct
or

Methods

Class
Name

Class

Data Members

Method 1
Expressions & Statements
Method 2
Expressions &
Statements

Account

-acctCount:int
-accountNo:int
-accountBalance:double
-accountType:String
+Account(String)
+depositAmount(double):void
+getAccountDetails():String



INSTANCE VARIABLE

- Stores the state of objects
- Declared outside methods and statement blocks in a class
- Can be directly accessed in member methods/blocks within a class

```
Syntax
<modifier> <type> <identifier> [= <initial value>];

private int accountNo;
private String customerName;
```

★ Note

- Instance variables are initialized to default values of their data types during object creation
- Every object has its own copy of instance variables in memory



STATIC VARIABLE

- Used to store common property shared by all objects of a class
- Only one copy of static variable is created in memory
- Belongs to the class and not an object

```
private static int acctCount;  
private static final double TAX_RATE = 9.0;
```

★ Note

- Static variables are created and initialized only once when the class is first loaded



CONSTRUCTOR

- Special method used for initialization of object
 - i.e. giving initial values to instance variables
- Has the same name as the class
- Does not return any values
- gets invoked automatically when an object is created

```
//NO ARGUMENT CONSTRUCTOR

public class Account{
    public Account() {
        accountNo = ++acctCount;
        accountBalance = 0;
    }
}
```



PARAMETERIZED CONSTRUCTOR

- During object creation, values can be assigned to instance variables, by passing parameters to the constructor

```
//PARAMETERIZED CONSTRUCTOR
public Account(String accType,double accBalance) {
    accountNo = ++acctCount;
    accountBalance = accBalance;
    accountType = accType;
}
```

★ Note

- Java provides the **default constructor**, If no constructor is provided by

```
public Account() { super(); }
```

- If programmer codes a constructor, Java doesn't provide default constructor

© 2014 Manipal Global Education Services



METHOD DECLARATION

```
Access          Return      Method          Parameters
Modifier       type        Name           rs
public double calculateAnswer(double param1, int[ ] param2){
    //Code method functionality here
    return x;
}
```

Method body

- Method represents the behavior of objects
 - Are identified by a method name which is unique in a class
 - Used to define object functionalities
 - Accept input parameters and return result
- Method needs to be invoked using a object reference, to execute their functionality



METHOD PARAMETERS

Primitive Reference

```
public double calculateAnswer(double param1, int[] param2){  
    //apply logic/do calculation here  
    return x;  
}
```

➤ Parameters

- Are the inputs provided to the method during execution
 - Declared in parentheses after method name
 - Every parameter has a datatype and a name
- Method can take zero or multiple parameters
- Parameters can be of **primitive or reference** data type
- Parameter values are passed, when the method is called/invoked



METHOD RETURN TYPE

```
public double calculateAnswer(double param1, int[] param2){  
    //apply logic/do calculation here  
    return x;  
}
```

Return type

Return Statement

- Methods can return back values as output using **return** statement
- During method declaration, return type defines the datatype of variable the method returns
- Return type can be primitive data type or reference data type
- Methods not returning anything should declare return type as “void”
- Any value that can be implicitly converted to the return type can be returned



METHOD BODY

```
public double calculateAnswer(double param1, int[] param2){  
    //apply logic/do calculation here  
    return x;  
}
```

- Method Body is written within curly braces after parameter declaration
- Functionality of the method i.e. business logic/calculation is coded here
- Method can access instance variables in the body
- One method can directly invoke another method of the same class without explicit reference



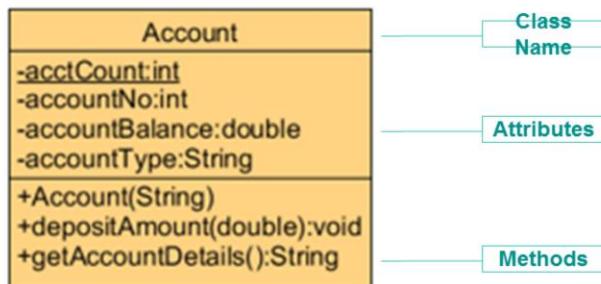
LOCAL VARIABLE

- Variables declared within the method body
- Cannot be accessed outside the method
- Have to be compulsorily initialized before using, to avoid compilation failure

```
public String getAccountDetails() {  
    String format = "";  
    format = "Acct No : " + accountNo;  
    return format;  
}
```



CLASS REPRESENTATION USING UML





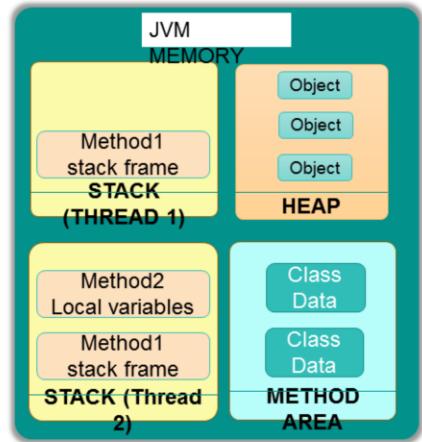
CONCEPT

Object creation



JVM Memory Model

- JVM organizes the memory it needs to execute a program into several runtime data areas
 - Heap
 - Memory area where objects are created
 - Shared by all threads
 - Stack
 - Stores the state of Java method invocations for a thread
 - Every thread has a separate stack
 - Method Area
 - Contains data about the class parsed from the



©2015 Manipal Global Education

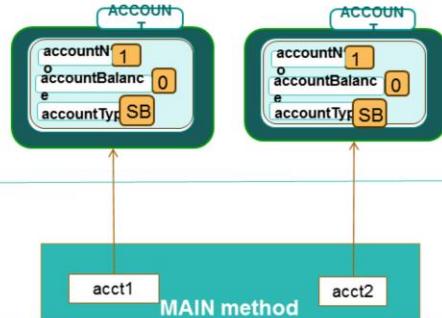


OBJECT CREATION

```
Account
acctCount:int
accountNo:int
accountBalance:double
accountType:String
+Account(String)
+depositAmount(double):void
+getAccountDetails():String
```

HEAP

STAC
K



```
class TestAccount {
    public static void main(String[] args) {
        Account acct1 = new Account();
        Account acct2 = new Account();
    }
}
```

Custom
Datatype

Reference
variable

new operator creates an
object on Heap and
returns its address



ACCESSING DATA MEMBERS

- Reference variable is used to access objects instance variable's and to invoke methods
- ~~Reference variable should be of same type as the Class of the object~~

```
Account acct1 = new Account();
```

- Access instance variable using dot notation

```
referencevariable.instancevariable  
int accNo = acct1.accountNo;  
System.out.println("Acct Type " + acct1.accountType);
```



Note

- instance variables can be directly accessed within the methods of the same class without explicit reference

20



INVOKING METHODS

- Methods are invoked using reference variable with a dot notation

```
referencevariable.method()
```

- Invocation of Account class methods in Test Class

```
Account acct1 = new Account();  
double depAmt = 1000;  
acct1.depositAmount(depAmt);
```

- Invocation of Method returning String

```
String details = acct1.getAccountDetails();
```

```
public class Account {  
    int accountNo;  
    double accountBalance;  
  
    public void depositAmount(double amount) {  
        accountBalance += amount;  
    }  
  
    public String getAccountDetails() {  
        String format = "Acct No :" + accountNo + "\n"  
                    + "Acct Balance: " + accountBalance;  
        return format;  
    }  
}
```



INVOKING METHODS

```
double depAmt = 1000;  
acct1.depositAmount (depAmt);
```

- When depositAmount(..) method is called, the value of depAmt (i.e. 1000) is copied in to parameter variable amount

Note

- Methods can directly invoke other methods in the same class without a explicit reference variable
- Method in classA can invoke a method of ClassB, using a reference of ClassB object

DEMO
Class : Account,
AccountTest

```
public class Account {  
    ...  
    public void depositAmount(double amount) {  
        accountBalance += amount;  
    }  
    ...
```



ACCESSING STATIC VARIABLES

- Static variables can be accessed using a Class Name with a dot notation

`ClassName.staticVariable`

```
int counter = Account.accountCount;
```

- Static variables can be accessed within and outside the defined class (Non private)

- Static variables can be accessed outside the defined class (Non private) but it is not recommended

```
Account acct1 = new Account();
int counter = acct1.accountCount;
```

Compiler provides a warning for the above access



STATIC METHODS

DEMO

Class : StaticDemo,
StaticTest

- Methods having static keyword in method declaration

```
public static String generateAccNo () {  
    return "A" + acctCounter++;  
}
```

- Can be invoked using a Class name as shown below
- Do not have access to instance variables

```
Account.generateAccNo();
```

- Why Static methods

- Common utility functions such as addition of numbers, sorting of arrays etc. are not dependent on state of objects
- Static methods make it possible to invoke such functionalities without a object instance



VARIABLE SCOPES

- Instance Variables
 - Stored in Heap memory along with the object
 - Accessible as long as the object is in heap and is referenced
- Static variables
 - Stored in method area memory along with the class object
 - Accessible as long as the class is loaded in memory
- Local variable
 - Stored in stack of the thread which is executing the method
 - Accessible as long as the method in which the variable is declared is executing



'this' REFERENCE

- “this” keyword is used to refer to the current executing object

```
class Employee {  
    String empName;  
  
    Employee(String empName) {  
        empName = empName;  
    } }
```

```
Employee emp = new Employee("John");  
System.out.println(emp.empName);
```

Output : null

- In the above example, since empName local variable shadows the instance variable, the instance variable will not be assigned the value “John”

```
Employee(String empName) {  
    this.empName = empName;  
}
```

- Can also be used for better readability

©2015 Manipal Global Education



1. _____ runs automatically when a new object of a class is created

2. For a given class, there is only one copy of a given _____ variable

3. If a method parameter accepts long, which all data types can be passed as arguments?

4. A object is accessible in a program as long as its _____ is available





Given

```
class Util{  
    int calcArea(int height, int width) {  
        return height*width;  
    }  
}
```

Consider "ut" as a reference of Util object. Which method invocations are valid ?

1. int a = ut.calcArea(7,12);
2. short c = 7; ut.calcArea(c,15);
3. int d = ut.calcArea(57);
4. long t = 42; int f = ut.calcArea(t,17);
5. byte h = ut.calcArea(4,20);





A special type of method that is used to initialize an object is called _____

- Constructor
- Destructor
- Modification
- editor





CONCEPT

Overloading



METHOD OVERLOADING

- Concept of defining more than one method with same name and different parameter list
 - Done for reusing method name for similar functionalities with different inputs
 - Ex: searchEmployee by Name, searchEmployee by EmpId
 - Method call is linked to appropriate method during compile time based on parameters – static polymorphism
 - Increases readability of the program
- Rules for Overloading
- Method Name should be same
 - Number of parameters should be different
 - Datatype of parameters should be different
 - Sequence of parameters should be different

31



METHOD OVERLOADING

```
public int add (int num1,int num2) {  
    return num1 + num2;  
}
```

Overloading by changing datatype

```
public float add (float num1,float num2) {  
    return num1 + num2;  
}
```

Overloading by changing Number of parameters

```
public int add (int num1, int num2, int num3) {  
    return num1 + num2 + num3;  
}
```



CONSTRUCTOR OVERLOADING

- Like methods, Constructors can also be overloaded in a class
- Constructor overloading provides the ability to have multiple constructors in a class with different parameters

```
public class Employee {  
    static int counter;  
    int empId;  
    String empName;  
  
    Employee() {  
        empId = ++counter;  
    }  
    Employee(String name) {  
        empId = ++counter;  
        empName = name;  
    } }  
  
Employee emp = new Employee();  
Employee emp = new Employee("John");
```

Overloaded
Constructor



EXPLICIT CONSTRUCTOR INVOCATION

DEMO
Class : Account,
AccountTest

- A constructor can explicitly invoke an overloaded constructor of the same class by using “this()”
- If present, this() should be the first statement in the constructor

```
public Account() {  
    accountNo = generateAccNo();  
}  
  
public Account(double amt){  
    this();  
    accountBalance = amt;  
}  
public Account(double amt, String opDate){  
    this(amt);  
    accountOpenDate = opDate;  
}
```

34

©2015 Manipal Global Education



What is output of the program?

```
class Test{
    public void display(int x, double y){
        System.out.println(x+y);
    }
    public double display(int p, double q){
        return (p+q);
    }
    public static void main(String args[]){
        Test test = new Test();
        test.display(4, 5.0);
        System.out.println(test.display(4, 5.0));
    }
}
```

- 9.0 9.0**
- 9 9**
- Compilation error**
- none**





CONCEPT

Initialization Block



INITIALIZATION BLOCK

- Initialization block is a block of statements inside curly braces

```
public class Employee {  
    {  
        //initialization code here  
    }  
}
```

- Executed when a object of the class is created
- Runs before the constructors are invoked
- Multiple initialization blocks can be present in a class and they will be executed in the sequence they are coded
- Typically used when some processing is required to initialize instance variable
- Can also be used to do common processing for all constructors in the class



STATIC INITIALIZATION BLOCK

```
public class Employee {  
    static {  
        //initialization code here  
    }  
}
```

- Executed when the class is first loaded
- Typically used to initialize static variables
- Do not have access to instance variables
- Class can have multiple initialization block



EXECUTION SEQUENCE

```
public class Employee {  
    private static int empCounter = 100;  
    private int empId = 0;  
  
    {  
        System.out.println("init block executed");  
    }  
  
    static{  
        System.out.println("static block executed");  
    }  
  
    public Employee(){  
        System.out.println("Constructor executed");  
    }  
  
    public static void main(String[] args) {  
        Employee e1 = new Employee();  
    }  
}
```

➤ Sequence of Execution

- Class Loaded
- Static variables initialized
- Static block executes
- Instance variables initialized
- Initialization block executes
- Constructor executes
- Object created



CONCEPT *Garbage Collection*

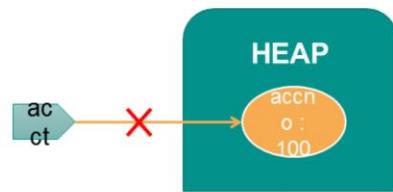


GARBAGE COLLECTION

- Garbage collection is the process of cleaning heap memory by removing unreferenced objects
- Done by a daemon thread called Garbage Collector
- Garbage collector runs automatically at frequent intervals based on memory availability
- Can be called explicitly using System.gc() but no guarantee that it will run

- When do object become garbage collectible ?

```
public class TestAccount{  
    public static void main(String[] args){  
        Account acct = new Account();  
        acct = null;  
    }  
}
```

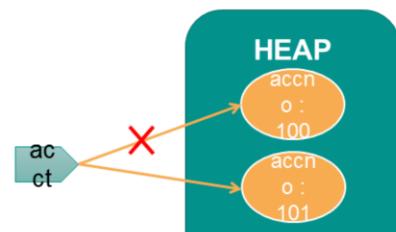




GARBAGE COLLECTION

- When do objects become garbage collectible ?
 - Reference reassigned

```
public class TestAccount{  
    public static void main(String[] args){  
        Account acct = new Account();  
        acct = new Account();  
    }  
}
```



- When objects have no references from stack after program execution completes

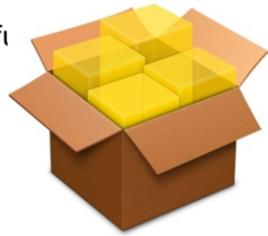


CONCEPT *Packages*



PACKAGES

- A package is a collection of related classes and interfaces in Java
- A group of related classes are bundled inside a package
- Java API is grouped in to multiple packages based on the functionality
 - java.lang : contains the fundamental classes
 - java.util : contains utility classes
- Why use packages
 - Avoids naming conflict
 - Provides access protection
 - Helps easily locate classes, interfaces and other types in the huge API
 - Helps in functionality wise grouping





SPECIFYING PACKAGES

- To put a class in the package, include the package statement with the package name as shown below

```
package com.banking;
public class Account{
    //code goes here
}
```

- package should be the first statement in a class/interfaces
- Fully qualified name of the class in a package is
<packagename>.<classname>
 - com.banking.Account
- Subpackages are packages created inside other packages and are treated as separate package
 - Ex. com.banking.loan is a sub-package of com.banking



IMPORTING PACKAGES

- To use a class of one package, in a class of another package, the class has to be imported using import statement

```
package com.banking;
public class Account{
    //code goes here
}
```

```
package com.test;
import com.banking.Account;
public class TestAccount{
    public static void main(String[] args) {
        Account acct = new Account();
    }
}
```

- There can be multiple import statements in a class for importing multiple classes/interfaces
- import statement should be written after package statement, if package is present



SOME JAVA API PACKAGES

package name	Description
java.lang	Contains classes that are fundamental to the design of the Java programming language
java.util	Contains utility classes like Calendar, Scanner, Date, Collection classes etc
java.io	Contains classes for system input and output through data streams, serialization and the file system
java.text	Contains classes and interfaces for handling text, dates, numbers, and messages
java.awt	Contains classes for creating GUI and for painting graphics and images
java.sql	Contains API for accessing and processing data stored in a data source (usually a relational database)

manipal Global Education



A package is a collection of

- Classes
- Interfaces
- editing tools
- classes and interfaces





In a java program, package declaration _____ import statements.

- must precede
- must succeed
- may precede or succeed
- none





package statement helps to create multiple classes to have _____ name.

- Differ
- Vanity
- Same
- punch





CONCEPT

Access Control

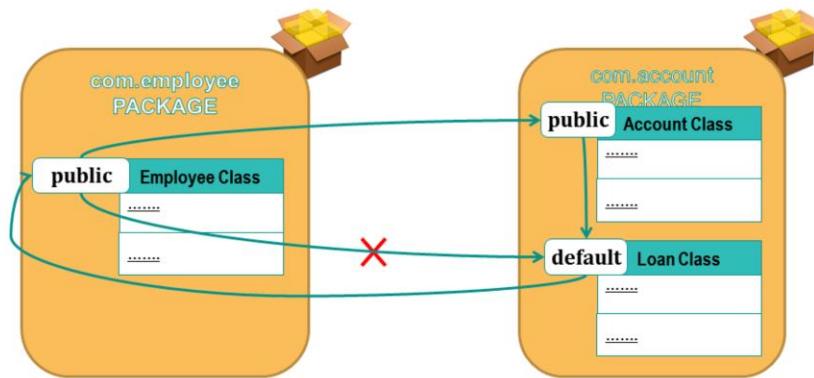


ACCESS MODIFIERS

- Used to expose or hide the attribute and behaviour of a class
- provide different access levels to the class/types and its members
- public (+)
 - declaration that is accessible to all classes
- private (-)
 - Declaration that is accessible only to the class/type in which it is declared
- default
 - Access to all classes/types within a package
 - Default declaration is done without specifying any modifier

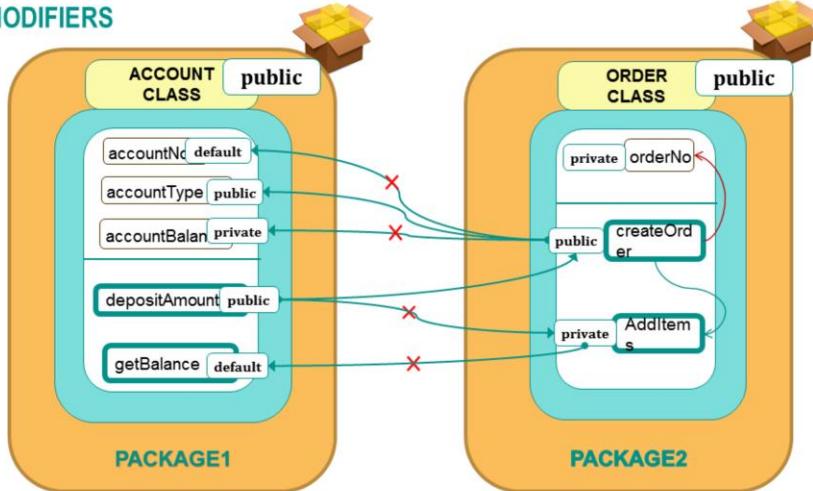


ACCESS MODIFIERS





ACCESS MODIFIERS





ACCESS MODIFIER EXAMPLE

```
public class Employee  
private instance variable  
empId  
public method getEmpId
```

```
public class Employee {  
    private int empId;  
    public int getEmpId() {  
        return empId;  
    } }
```

```
class Employee {  
    public static int empcounter = 1000;  
    private int empId;  
  
    int getEmpId() {  
        return empId;  
    } }
```

```
default class Employee  
public static variable  
empCounter  
private instance variable  
empId
```



```
class Access{
    public int x;
    private int y;
    void cal(int a, int b){ x = a + 1; y = b; }
}
class AccessSpecifier {
    public static void main(String args[]){
        Access obj = new Access();
        obj.cal(2, 3);
        System.out.println(obj.x + " " + obj.y);
    }
}
```

- 3 3
- 2 3
- Compilation error
- Runtime error





References

- Refer following demo videos on EduNxt
 - M4L1L3_Demonstrating_use_of_different_types_of_variables_in_a_class – Demo
 - M4L1L5_Demonstrating_methods_in_a_class – Demo
 - M4L1L7_Demonstrating_the_use_of_'this'_keyword – Demo
 - M4L2L2_Demonstrating_constructors – Demo
 - M4L2L4_Demonstrating_parameterized_constructors – Demo
 - M4L3L2_Static_class_members – Demo
 - M4I9I2 Demonstrating Package And Import - Demo





SUMMARY

Classes and Objects



SUMMARY

In this lesson, you've learned to:



- Create a class and Instantiate an object
- Access Data members and invoke methods
- Define static members and initialization blocks
- Use "this" reference
- Overload Methods and Constructors
- Use Packages
- Use Access Modifiers