* **Object**

Object is an instance of a class and represents a real-world entity.

Object Lifetime:

* + Persistent

Is stored in the databased and can be accessed after the process that created it terminates.

* + - Stored in the database.
    - Visible by all nodes.
  + Transient

Transient objects are not stored in the databased and are destroyed when deleted or after the process that created it terminates.

* + - Temporary objects
    - Only visible to the process
  + Shared Transient

Is visible through all processes within the same node, where each process in the node can access the object as it would access a persistent object. Is destroyed by deleting it or terminating all processes in the node.

* + - Temporary object
    - Visible to all processes in a node.
* **Class**

Describes common characteristics and behaviours of a set of objects.

* Instantiation
  + Abstract class cannot have instances but can be used to group similar subclasses that share some state and behaviour.
  + Real class can have instances.
  + Often a superclass is abstract.
* Inheritance

Defines an “is-a-kind-of” hierarchy among classes in which a subclass inherits properties and methods defined in one or more superclasses.

**Polymorphism**

The behaviour of an object at runtime depends on the class it belongs to. A message sent to an object results in the execution of a method, but the actual implementation of the method that is executed depends on the class of the object.

Polymorphism is the ability of an object to respond differently to a given message depending on its type.

* **Schema**

Schema is a collection of classes. It is the highest level organisation unit in JADE. It encapsulates a class hierarchy for a particular domain.

* **Map Files**

Map File is a contraction of ‘Class Map’ and ‘Database File’. Class Mapping is the process of mapping classes to the files in which the instances of the class are stored.

Map File must be added when creating a schema. You may create as many map files as you like.

* **Layout of class browser**

Class browser is divided into four window panes:

* Classes
* Properties
* Methods
* Editor
* **Structure of a Method**

A method consists of the following elements:

* Signature
* Constant Section – optional
* Vars section – optional
* Method Body
* Epilog section - optional
* **JadeScript Method**

Primitive values such as date, time, string, etc, are handled using primitive types rather than objects. A variable or attribute that is a primitive type contains a value as opposed to a reference to an object.

* **Features of JADE primitive types**

JADE primitive types have the following features:

* Have methods supplied by JADE. You can also add your won methods to primitive types.
* Cannot have properties.
* Can have constants defined for them.
* New primitive types cannot be defined.
* **Collection**

Is an object that stores either:

* Primitive types (for example, an Integer Array contains a series of integers), or
* References to other objects (for example, an ObjectSet contains a series of object references).

Whilst the size of an individual reference in a collection is small, the size of a collection object increases with the number of objects in the collection.

There are three main types of collections:

* + Set

An unordered collection of objects. A set cannot store primitive values.

* + Array

An ordered collection of objects of primitive values. The order of the entries within the array can be controlled by your code. Unlike sets or dictionaries, arrays can hold the same object or primitive value more than once.

* + Dictionary

Order the objects according to the keys that you define. A dictionary cannot store primitive values.

* **Iterating collections**

There are two ways to iterate a collection: Either use a foreach instruction, or an iterator.

A foreach iteration is often the most convenient option, but it is also less flexible than explicitly using an iterator object. Here is an example of a foreach iteration that displays the name of each product in a customer’s allProducts dictionary:

*foreach product in cust.allProducts do*

*write product.name;*

*endforeach;*

* **Iterator/while**

An iterator is an instance of the Iterator class. You must create an iterator object before using it to iterate a collection. The following method shows the use of an iterator:

*testIterator();*

*vars*

*coll : Collection;*

*cust : Customer;*

*iter : Iterator;*

*begin*

*// Reference the collection*

*coll := Root.firstInstance.allCustomers;*

*//Create the iterator*

*iter := coll.createIterator;*

*//Iterate through the collection and*

*// display each customer’s last name*

*while iter.next(cust) do*

*write cust.lastName;*

*endwhile;*

*epilog*

*delete iter;*

*end;*

* **Iteration control instructions**

There are two instructions that alter the normal flow of control during an iteration.

* The *break* instruction.
* The *continue* instruction.
* **Form**

Forms are the interface between the user and the developer defined database. They control how the application behaves so it is important that they are designed correctly.

* Form Methods and Events

Forms are just like other objects except they have an associated window, a physical displayable image. You must first create a transient instance of the form.

*vars*

*form : CustomerDetails;*

*begin*

*create form transient;*

As forms are always transient you do not need to be in transaction state.

Next, you use the show method to display the form.

*form.show();*

Before the form is displayed it executes any code in the load, paint, activate and finally *gotFocus* events.

To delete a form, use the *unloadForm* method of the form. The form is then automatically deleted as part of the tidy up process.

*self.unloadForm();*

Before the image disappears any code in the *queryUnload* and *unload* events is executed.

* **MDI**

Stands for Multiple Document Interface. It is a standard Windows term used to describe the behaviour of forms in a user application.

A form can be an MDI Frame, an MDI Child or neither. The value of mdiFrame and mdiChild determine the behaviour of the form.

* MDI Frame is a form used as a container for other forms that are MID children.
* MDI Child
* **Application-Related Objects**
* What is an application?

A GUI application enables a user to interact with the system through forms displayed on a screen. Other types of application can be used for background tasks and parallel processing. The application types are explained shortly. There may be many applications providing different views of the database.

* Application Object

When an application is started, JADE creates an application object, referred to as **app**, which is a transient instance of your schema application class. Your schema application is a subclass of the system **Application** class.

The Application class contain many features that are used to control the behaviour of your application, for example *showBubbleHelp*. It also provides many useful services to the locally running application, such as a method to display a message box.

When a schema includes imported packages, an additional application object is maintained for each schema from which a package has been imported. This ensures that methods from the schema in which a package is implemented can access features of the local application class.

* Process Object

An application runs in a thread of execution in the node, called a process. A persistent instance of the Process class, referred to as process, represents the process.

* Global Object

The persistent instance of your Global subclass, called global, provides logon security checking services to your application.

* Form Objects

When you design a form in the Painter, the definition of the form is stored in JADE database as a persistent instance of the Form class.

* **Application Types**

When you create an application in JADE you must specify the type of application it will be. There are five types of application supported by JADE.

* GUI (default)
* GUI, no forms
* Non GUI
* Web Enabled

There are three types of web applications:

* + Jade Forms
  + HTML Documents
  + Web Services
* Web-Enabled, Non GUI
* **Startup Form**

Is a form defined in the current schema that will be displayed when the application initializes.

* **Initialize Method**

When you define an application you may nominate a method, known as the initialize method, to be executed automatically when the application starts. It must be a method defined in the schema application class or one of its super classes.

* **Finalize Method**

When you define an application you may nominate a method, known as the finalize method, to be executed automatically when the application terminates. It must be a method defined in the schema application class or one of its super classes.

* **Why have a root object?**  
  A common design strategy is to add a class that has only one instance, which represents the business or organization that the software serves. The ‘root’ object is a convenient object to own the complete collections of instances of a class required by the application. Thus a customer may have a collection of his or her bank accounts, but if the application requires a complete collection of bank accounts belonging to all customers then the root object is a logical place to store it.
* **How is the Root Object Accessed?**

Because there is only one root object it can be accessed by the *firstInstance* method. However the firstInstance method has to be executed on the server, so it makes sense to keep a local reference to the root object on the client. The app object, a local transient object created when the application starts is ideal for this purpose.

* **Security Methods**
  + **User Validation Support**
  + **getAndValidateUser**
  + **isUserValid**