AvalonFramework

What is Avalon-Framework?

Avalon-Framework is a small but vital jar file distributed by the Avalon project, and some associated documentation. Avalon-Framework (often abbreviated as AF, or even AF4 for version 4 of the framework) defines several "lifecycle interfaces" that provide a common way for components to be created, initialized, configured, started, etcetera. Part of that lifecycle is support for Inversion of Control. With a fancy term, Avalon provides service-locator-based-dependency-injection. In less fancy terms, Avalon-Framework components implement an interface, Serviceable:

```
public interface Serviceable {
  public void service(ServiceManager sm) throws ServiceException;
}
```

After instantiation of an Avalon-Framework component, this method is called in order to make the component aware of its environment. It will use this provided ServiceManager to gain access to the other components it needs. The ServiceManager is again a relatively simple interface:

```
public interface ServiceManager {
  boolean hasService(String key);
  Object lookup(String key) throws ServiceException;
  void release(Object object);
}
```

there are several other lifecycle interface, but this one is probably the most important. Here's an example of two Avalon-Framework components:

```
/**
 * This kind of interface is often referred to as a
 * "work interface". A component which implements this
 * interface can do everything a cat can do.
 *
 * An example of an interface that is not a work
 * interface would be java.io.Serializable, or the
 * above Serviceable interface. Implementing Serviceable
 * means a component uses other components, but it doesn't
 * specify anything about what the component does.
 */
public interface Mouse {
  int runReallyFast();
}
/**
 * It might not look like it, but this is an Avalon-Framework
 * component! Why? Because it has a public constructor that
 * doesn't take any arguments and implements all other
 * Avalon-Framework rules.
 */
public class LittleMouse implements Mouse {
  public LittleMouse() {}
  public int runReallyFast() {
    return (int)((Math.random()*12)+1);
  }
}

/**
 * Another work interface!
 */
public interface Cat {
  public void hunt();
}
/**
 * And another component! We say that VeryFastCat has
 * a dependency on another component, in this case, on
 * an implementation of Mouse.
 */
public class VeryFastCat implements Serviceable {
  private Mouse m_prey;
  private int speed = 10;

  public VeryFastCat() {}

  public void service(ServiceManager sm)
      throws ServiceException {
    m_prey = sm.lookup(Mouse.class.getName());
  }

  public void hunt() {
    if(m_prey.runRealFast() =< speed)
      System.out.println( "Caught the mouse!" );
    else
      System.out.println( "The mouse got away!" );
  }
}
```

But who creates the ServiceManager? Who sets all this up?

You may have noticed that the sample above is missing a few vital pieces. There's no implementation of ServiceManager to use, for example. And there's also no code that creates instances of LittleMouse and VeryFastCat. That code might look like this:

```
public void testCreateTheComponentsManually() throws Exception
{
  DefaultServiceManager sm = new DefaultServiceManager();
    // DefaultServiceManager is also part of Avalon-Framework....
  LittleMouse mouse = new LittleMouse();
  VeryFastCat cat = new VeryFastCat();
  sm.put( Mouse.class.getName(), mouse );
  sm.put( VeryFastCat.getName(), cat );
  sm.makeReadOnly();

  cat.service( sm );
  cat.hunt(); // prints something...
}
```

Now, it doesn't seem like Avalon-Framework really got us anything. Which is true. Avalon-Framework by itself doesn't do a whole lot. The big advantage with Avalon-Framework components is that you don't need to write the above code! Instead, you use a *container*, such as Excalibur's fortress container, to do all this for you.

That's very useful indeed, which is why many software projects utilize Avalon-Framework.

There's of course a lot more to it, but these are the basic basics. For more information about Avalon-Framework, see the Framework section of the Excalibur website.