Explanation of Analysis101 Driver

Explanation of the Analysis101 Driver

Explanation of Driver Code

Below is the complete text of the **Analysis101** Driver, stripped of comments.

```
import org.lcsim.util.aida.AIDA;
import hep.physics.vec.VecOp;
import java.util.List;
import org.lcsim.event.EventHeader;
import org.lcsim.event.MCParticle;
import org.lcsim.util.Driver;

public class Analysis101 extends Driver
{
   private AIDA aida = AIDA.defaultInstance();

   protected void process(EventHeader event)
   {
      List<MCParticle> particles = event.get(MCParticle.class,event.MC_PARTICLES);

      aida.cloud1D("nTracks").fill(particles.size());

      for (MCParticle particle : particles)
      {
         aida.cloud1D("energy").fill(particle.getEnergy());
         aida.cloud1D("cosTheta").fill(VecOp.cosTheta(particle.getMomentum()));
         aida.cloud1D("phi").fill(VecOp.phi(particle.getMomentum()));
      }
   }
}
```

First, as in all Java programs, there are the import statements.

```
import org.lcsim.util.aida.AIDA;
import hep.physics.vec.VecOp;
import java.util.List;
import org.lcsim.event.EventHeader;
import org.lcsim.event.MCParticle;
import org.lcsim.util.Driver;
```

This includes code libraries into the Driver that are needed to do the analysis.

For instance, methods of **org.lcsim.util.aida.AIDA** are used to book and fill the plots.

The **hep.physics.vec.VecOp** package is for doing math operations on vectors.

Next is the declaration of the Driver class.

```
public class Analysis101 extends Driver
```

The **Analysis101** class extends Driver, which means that it can override some or all of Driver's public or protected methods to do some useful work, like filling histograms.

The class stores a reference to the default AIDA object.

```
private AIDA aida = AIDA.defaultInstance();
```

This is convenient for creating and filling histograms "on-the-fly".

The single function in this class is called process. Analysis101 inherits this method from Driver.

```
protected void process(EventHeader event)
```

Its single argument is the EventHeader of the current LCIO event. All collections in this event are accessible through the EventHeader interface.

> **EventHeader API**
>
> Refer to the EventHeader JavaDoc for information on all the available methods in this interface.

This line retrieves a list of the MCParticles in the event.

```
List<MCParticle> particles = event.get(MCParticle.class,event.MC_PARTICLES);
```

This uses the get method to retrieve all collections of type (class) MCParticle. The

```
event.MC_PARTICLES
```

constant specifies the name of the (sub)collection to retrieve, so that we get back a list of objects (rather than a list of collections, e.g. a list of lists).

Now that we have a list of particles in this event, some plots can be filled.

This line fills a plot called "nTracks" with the number of particles in the event.

```
aida.cloud1D("nTracks").fill(particles.size());
```

For convenience, the cloud1D method of the AIDA object will create the "nTracks" plot if it doesn't already exist. Otherwise, the existing plot will be filled.

The MCParticles List is a generic Java collection, so its size is conveniently available using the **size()** method.

To fill some plots with data from individual particles, the Driver loops over the list using Java's "foreach" construct.

```
for (MCParticle particle : particles)
```

This means that the body of the loop can do processing on each particle in **particles** using the **particle** variable.

Next, three plots are filled with data from the particle.

```
aida.cloud1D("energy").fill(particle.getEnergy());
aida.cloud1D("cosTheta").fill(VecOp.cosTheta(particle.getMomentum()));
aida.cloud1D("phi").fill(VecOp.phi(particle.getMomentum()));
```

The first line plots the particle's energy. The second two characterize the particle's momentum/direction by using the cosTheta and phi vector operations on the momentum value
retrieved from the particle.

> **MCParticle Interface**
>
> Refer to the MCParticle JavaDoc for a complete list of all the methods available in this interface. MCParticle inherits some of its functionality from its parent interface, **Particle**.