JunctionDispatcherDealer, ChannelSwimmerDealer, JunctionSwimmerDealer etc. will carry a map which stores a Junction object as a key and (a particle type and a Dispatcher object pair map as a value (in case of JunctionDispatcherDealer) or a channel as a key and (a particle type and a Swimmer pair map) as a value (in case of ChannelSwimmerDealer). In case of JunctionDispatcherDealer, If there are multiple calculations corresponding to different conditions (e.g., flow conditions) in a junction, the calculations should be dealt with in the particular dispatcher.dispatch(particle). Maybe a common route calculation (e.g., the calculation is only depending on flow split) can be as a class method in JunctionDispatcherDealer class?

The Dealer classes stated above are more like an observer. They look over particle and decided if an adequate behavior (dispatcher, swimmer, etc) need to be attached to the particle.

These Dealer classes need only one object and are passed to a particle as something similar to Observer in PTM code: setObserverForParticle in MainPTM line 131, installObserver in ParticleObserver line60, installObserver in Particle line 198, observeChange in Particle line 682.

Public class JunctionDispatcherDealer{

Private Map jdmap;

public JunctiondispatcherDealer(){// don’t need to do anything or set internal nodes for junctions?}

public add(Junction, particle type, Dispatcher){…}

public lookup(Junction, particle type){

if found, in the map return a pair of junction, dispatcher

if no, return pair of general junction and dispatcher

}

public dispatch(particle){

1. call look up to get pair of Junction, dispatcher (junction, dispatcher)

// the junction could be GSJunction for Salmon, e.g., GSJSDispatcher

1. do the accept and dispatch:

junction.accept(dispatcher, particle);

e.g., GSJunction.accept(GSJSDispatcher, particle);

// not using

1. Search map using a internal node number (implement a match method in the dispatcher interface) to find a match junction
2. If yes Call dispatcher
3. if not using default (the class static method)
4. //

}

}

Public interface Junction{

Public void accept(JunctionDispatcher, Particle);

}

**GSJunction should be a singleton? Only include node and barrer op info. Node will be a PTM node and barrier op info will be a map which will have waterbody as a key and a timeseries hashmap as value.**

**class** GSJunction **implements** Junction {

private node \_node = //GS PTM node

// map indexed by waterbody numbers

private Map<Integer, NonPhysicalBarrier> \_barriers;

**public** Junction(Node node) {

**\_node** = node;

\_barriers = null;

}

**public** Junction(Node node, Map<Integer, NonPhysicalBarrier>, barriers) {

**\_node** = node;

\_barriers = barriers;

}

**public** Node getNode() {

**return** **\_node**;

}

public map<period, integer> getBarrier(wbNumber){ return \_barriers(wbNumber); }

public setNode;

public setBarriers;

**public** **void** accept(JunctionDispatcher jdp, Particle p) { // was thinging to use

GSJDispatcher as an extended interface of all GSJunction dispatchers so that it’ll

call a GSJ dispatcher not something else, but then I have to implement all the

different accept that take different classes

*/\**

*\* accept(JunctionDispatcher) in GSJunction implements*

*\* accept(JunctionDispatcher) in Junction, so the call*

*\* to accept is bound at run time. This can be considered*

*\* the first dispatch. However, the decision to call*

*\* dispatch(GSJunction, Particle) (as opposed to dispatch(HOJunction, Particle)*

*etc.) can be*

*\* made during compile time since 'this' is known at compile*

*\* time to be a GSJunction. Moreover, each implementation of*

*\* Dispatcher implements the dispatch(GSJunction, Particle), which is*

*\* another decision that is made at run time. This can be*

*\* considered the second dispatch.*

*\*/*

// do I have to cast the jdp type? No, Otherwise how it would know which dispatch

to Call? the type of jdp will be known at run time?

jdp.dispatch(**this,p**);

}

}

**class** HOJunction **implements** Junction {

**public** **void** accept(JunctionDispatcher jdp, Particle p) {

jdp.dispatch(**this, p**);

}

}

**/\* need this? will this creat error since this type is universal, not safe? But if this class used, the implementation will be separated from the dealer thus make it easier to maintain.\*/**

**class** AJunction **implements** Junction {

**public** **void** accept(JunctionDispatch jdp, Particle p) {

jdp.dispatch(**this, p**);

}

}

Public interface JunctionDispacher{

Public void dispatch(Junction, Particle);

In the implementation a particle method that updates the particle current waterbody should be called.

// not using

Void dispatch(GSJunction gsj);

Void dispatch (HOJunction hoj);

Void dispatch (AJunction aj);

Do we need to have those different dispatches or only one Junction interface:

Void dispatch(Junction);

//

}

Public class GSJSalDispatcher{

Public void dispatch(Junction, Particle){

// here Junction need to be cast to GSJunction and should be able to catch the exception of cast error

// Particle type need to be check for Salmon

}

}

Public class GSJSteelDispatcher{

Public void dispatch(Junction, Particle){

// here Junction need to be cast to GSJunction and should be able to catch the exception of cast error

// Particle type need to be check for steelhead

}

}

Public class HOJSalDispatcher{

Public void dispatch(Junction, Particle){

// here Junction need to be cast to HOJunction and should be able to catch the exception of cast error

// Particle type need to be check for Salmon

}

}

Is this dangerous to not specify any particular type junction, because the error will be only caught in the run time if a run junction/dispatcher passed in. No if e.g., junction is HO but dispatcher is GS it will catch the class cast error at run time.