Use the ***Builder Pattern*** to encapsulate the construction of a product and allow it to be constructed in steps.

***Scenario – Vacation Planner:***

Vacation

Day Two

Day One

Hotel Park Tickets Dining Hotel Park Tickets Special Event Dining

Dinner

Dinner Patterns On Ice

VacationBuilder

Vacation

buildDay()

addHotel()

addReservation()

addSpecialEvent()

addTickets()

getVacationPlanner()

AbstractBuilder

buildDay()

addHotel()

addReservation()

addSpecialEvent()

addTickets()

getVacationPlanner()

builder.buildDay(date);

builder.addHotel(date, “Sheraton”);

builder.addTickets(“Patterns On Ice”);

// plan rest of vacation

Planner yourPlanner = builder.getVacationPlanner();

Client

constructPlanner()

***Benefits***:

* Encapsulates the way a complex object is constructed.
* Allows objects to be constructed in a multistep and varying process (as opposed to one-step factories)
* Hides the internal representation of the product from the client.
* Product implementations can be swapped in and out because the client only sees an abstract interface.

***Uses and Drawbacks***:

* Often used for building composite structures
* Constructing objects requires more domain knowledge of the client than when using a factory.