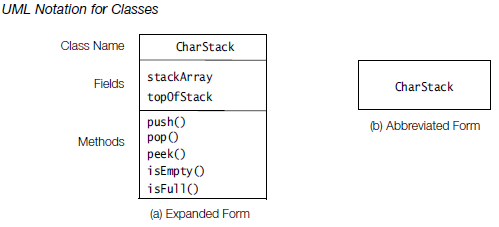
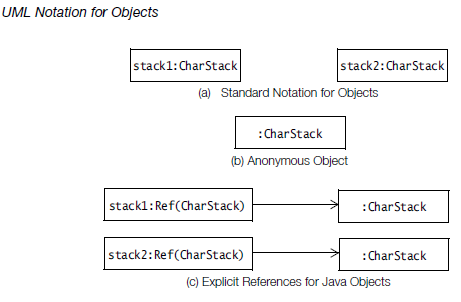
**Class Diagram**

Using Unified Modeling Language (UML) notation, a class called CharStack

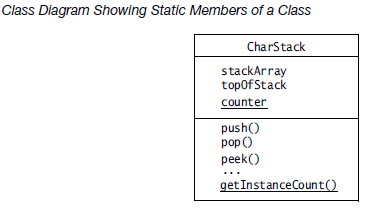
is graphically depicted



******

the class diagram for the class CharStack. It has been augmented

by two static members that are shown underlined.

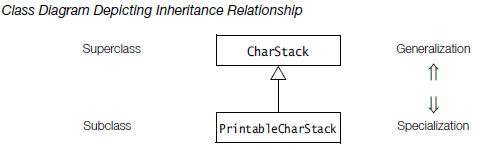
******

The class

PrintableCharStack is called the *subclass*, and the class CharStack is called the *superclass*.

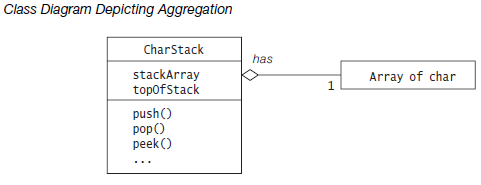
The CharStack class is a *generalization* for all stacks of characters, whereas the class PrintableCharStack is a *specialization* of stacks of characters that can also print

their elements.

******

Java supports aggregation of objects by reference, since objects cannot contain

other objects explicitly.

******

***Identifying the Boundary***

It is important to clearly define the boundary of your system. Things inside the boundary

of the system are things you need to worry about creating. In a UML use case diagram,

the *system boundary* is denoted by a rectangle, as in Figure 3.

***use case***

A *use case* is a *specification of sequences of actions, including variant sequences and*

*error sequences, that a system, subsystem, or class can perform by interacting with*

*outside actors*

In UML, a use case is represented by an oval

It is best to express your use case title/label in a few words (generally no more than five

words). These few words must begin with a present-tense verb phrase in active voice,

stating the action that must take place (notice: **Draw** Card, **Get Out** of Jail, and **Switch**

Turn).

***Identifying the Use Cases***

The following questions can be asked to identify use cases, once your actors have been identified

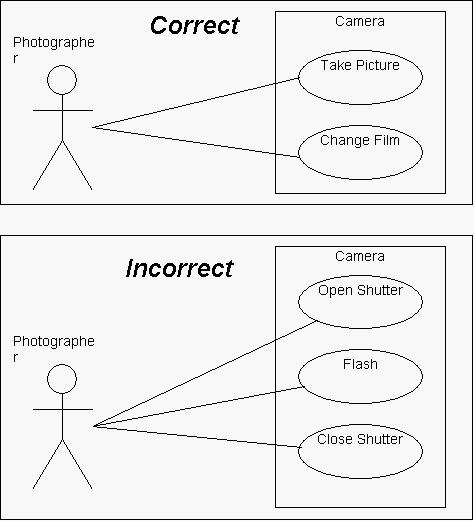
• What functions will the actor want from the system?

• Does the system store information? What actors will create, read, update or delete this information?

• Does the system need to notify an actor about chances in the internal state?

• Are there any external events the system must know about? What actor informs the system of those events?

In the diagram below we would like to represent the use cases for a camera. Suppose we choose "Open Shutter", "Flash", and "Close Shutter" as the top-level use cases. Certainly these are all behaviors that a camera has, but no photographer would ever pick up their camera, open the shutter, and then put it down, satisfied with their photographic session for the day. The crucial thing to realize is that these behaviors are not done in isolation, but are rather a part of a more high-level use case, "Take Picture". (Note that it does make sense for a photographer to "Take Picture" just once during a session with their camera.)



***Scenarios***

Scenarios are a set of scenarios tied

together by a common user goal (Fowler, 2000) or a sequence of transactions performed

by a system that yields an outwardly visible, measurable result of value for a particular

actor.

***Note: Use Case Is Made Up of Scenarios***

***Actor***

An *actor* is *an entity that interacts with the system and/or needs to exchange information*

*with the system*. The actor is *not* part of the system itself and should be included to

represent anyone or anything that interacts with the system in the following ways:

• supplies input information to the system

• receives information from the system

• both supplies input information to and receives information from the system

In UML symbols, an actor is represented as a stickman

An actor is always a noun in the scenario.

You should think of the actors as roles, not as “individuals.”

For example, you might know that several players will play the game. However, they would all be represented by one actor because they all have the same role.

If you happen to know that one person might take on several roles, such as player and bad player, you might be tempted to combine those roles into one actor.

***Identifying the Actors***

The following questions can help you identify the actors of your system

• Who uses the system?

• Who installs the system?

• Who starts up the system?

• Who maintains the system?

• Who shuts down the system?

• What other systems use this system?

• Who gets information from this system?

• Who provides information to the system?

• Does anything happen automatically at a present time?

***Use Case Diagram***

UML Use Case Diagrams can be used to describe the functionality of a system in a horizontal way.

A *use case diagram* is a visual representation of the relationships between actors and use cases together that documents the system’s intended behaviour.

Arrows and lines are drawn between actors and use cases and between use cases to show their relationships.

UCDs have only 4 major elements: The **actors** that the system you are describing interacts with, the **system** itself, the **use cases**, or services, that the system knows how to perform, and the lines that represent **relationships** between these elements.

Refrences:

<http://agile.csc.ncsu.edu/SEMaterials/UseCaseRequirements.pdf>

<http://www.andrew.cmu.edu/course/90-754/umlucdfaq.html>

Draw the User Case Diagram online by using the below site

[http://www.gliffy.com/gliffy/#](http://www.gliffy.com/gliffy/)

[suraj06k@gmail.com](mailto:suraj06k@gmail.com) (L……….)

**UML Sequence Diagrams**

UML sequence diagrams are used to show how objects interact in a given situation. An important characteristic of a sequence diagram is that time passes from top to bottom : the interaction starts near the top of the diagram and ends at the bottom (i.e. **L**ower equals **L**ater).

As with all UML diagrams, comments are shown in a rectangle with a folded-over corner :

UML sequence diagram with a comment.

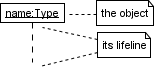
**Targets**

Objects as well as classes can be targets on a sequence diagram, which means that messages can be sent to them. A target is displayed as a rectangle with some text in it.

The lifeline is displayed as a vertical dashed line.

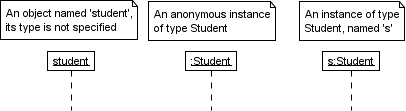
Object

The basic notation for an object is

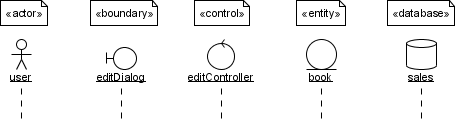


Where 'name' is the name of the object in the context of the diagram and 'Type' indicates the type of which the object is an instance. Note that the object doesn't have to be a *direct* instance of Type, a type of which it is an *indirect* instance is possible too. So 'Type' can be an abstract type as well.

Both name and type are optional, but at least one of them should be present. Some example :



As with any UML-element, you can add a stereotype to a target. Some often used stereotypes for objects are «actor», «boundary», «control», «entity» and «database». They can be displayed with icons as well :



**Class**

The basic notation for a class is

UML sequence diagram of a class.

Only class messages (e.g. shared or static methods in some programming languages) can be sent to a class. Note that the text of a class is not underlined, which is how you can distinguish it from an object.

Class Diagram

**Class Diagram Example: Element Visibility**

|  |  |
| --- | --- |
| /\*\* \* Attribute and operation visility \* UML User Guide p. 123 \* \* @opt operations \* @opt attributes \* @opt types \* @opt visibility \* @hidden \*/ **class** UMLOptions {}  /\*\* @hidden \*/ **class** Tool {}  **class** Toolbar { **protected** Tool currentSelection; **protected** Integer toolCount; **public** **void** pickItem(Integer i) {} **public** **void** addTool(Tool t) {} **public** **void** removeTool(Integer i) {} **public** Tool getTool() {} **protected** **void** checkOrphans() {} **private** **void** compact() {} } | UML diagram showing attribute and operation visility |