

ONTOS Programming.

Open and Close

```
OC_Boolean OC_open(char* dbname);  
void OC_close(char* dbname=0,  
              OC_Boolean cleanAll=FALSE);
```

Transaction

```
void OC_transactionStart();  
void OC_transactionAbort();  
void OC_transactionCommit();
```

Example;

```
char *dbname = "c53DB";
if (!OC_open (dbname) ) {
    // ...
}
// the conservative policy
OC_transactionStart ();
// application program
OC_transactionCommit ();
OC_close ();
```

Persistent Object: ONTOS DB places the following additional requirement on class definitions.

- Persistent classes must be directly or indirectly from class "OC_Object".
- an APL(Activation Parameter List) constructor must be defined for each class.
- Define a getDirectType() member function.
 - When ONTOS DB needs to access the type of an entity, it calls this function. This function can not be defined in-line.
- Include a call to initDirectType() in the new instance constructor. It must be called if the object's type has an extension of all of its instances.
- If a persistent class requires a destructor, the class definer must also supply a void function called "Destroy()" and should move the destructor code to the Destroy function instead.

Example: A Class Definition

```
class Person : public OC_Object {
    int age;
public:
    Person(int its_age, char *its_name=NULL);
    Person(APL*);
    OC_Type *getDirectType();
    int age() {return priv_age;}
    void age(int its_age){age = its_age;}
    char *name();
    void name(char* its_name);
};

Person::Person(APL *theAPL):OC_Object(theAPL){}
OC_Type *Person::getDirectType()
{
    return (OC_Type*)OC_Lookup("Person");
}
```

In any constructor except the APL constructor, you must add additional lines as follows.

```
Person::Person(int its_age, char* its_name)
    : OC_Object (its_name)
{
    OC_Type *PersonType;

    PersonType = getDirectType();
    initDirectType(PersonType);
    // ...
}
```

Creation of objects

```
Person *p1 = new Person(20, "Mr.Right");
```

How to assign objects a name?

```
Person::Person(int its_age, char* its_name)  
    : oc_object(it's_name) { /* ... */ }
```

```
void Person::name(char* the_name)  
{  
    Name(the_name) ;  
}
```

How to return an object name?

```
char* Person::name()  
{  
    return Name();  
}
```

How to store an object into a database?

Deactivation is the term used for writing objects back to an database.

```
Person *p0, *p1;
OC_transactionStart();
p0 = new Person(30, "Mr. Brown");
p1 = new Person(20, "Mr. Right");
p1.putObject();
OC_transactionCommit();
```

How to get an object with its name?

Activation is used for reading objects from a database.

```
Person *p2;
OC_transactionStart();
p2 = (Person*)OC_lookup("Mr. Right");
cout << p2.name();
OC_transactionCommit();
```

How to delete objects from a database?

```
Person *p2, *p3;

OC_transactionStart();
p2 = (Person*)OC_lookup("Mr.Right");
p3 = (Person*)OC_lookup("Mr.Smith");
delete p2;
p3->deleteObject();
OC_transactionCommit();
```

More about Object

```
class OC_Object : public OC_Entity {
protected:
    OC_Object(char* name=0);
public:
    OC_Object(OC_APL* the APL);
    ~OC_Object();
    virtual void Name(char *newName);
    virtual char* Name();
    void* operator new(OC_size_t sz);
    void* operator new(OC_size_t sz, StorateManager* sm);
    void* operator new(OC_size_t sz, Object* where,
        Clustering howNear); // total six new operators
    void operator delete(void*);
    virtual void putObject(OC_Boolean deallocate=OC_false);
    virtual void deleteObject(OC_Boolean deallocate=OC_true);
    OC_Boolean isNamed();
    OC_Boolean isNewObject();
    OC_Boolean isObjectInDB();
    virtual OC_LockType getLockType();
    virtual void lockObject(OC_LockType lock);
    // ...
};
```

More about Entity

```
class OC_Entity : public OC_CleanupObj {
protected:
    OC_Entity();
    virtual void initDirectType(Type* theType);
public:
    virtual OC_Type* getDirectType()=0;
    virtual OC_Boolean Equal(Entity* another);
    virtual OC_Boolean operator==(Entity& another);
    virtual OC_Boolean operator>(Entity& another);
    virtual OC_Boolean operator<(Entity& another);
    virtual OC_Boolean operator<=(Entity& another);
    virtual OC_Boolean operator!=(Entity& another);
    OC_Entity(APL* theAPL);
    virtual unsigned long Size();
    // ...
};
```

Include a destructor() if there is a destructor.

```
class Person : public OC_Object {
    int age;
public:
    Person(int its_age, char *its_name=NULL);
    Person(OC_APL*);
    ~Person(){destroy(OC_false)};
    OC_Type *getDirectType();
    void destroy(OC_Boolean aborted=OC_false);
};

class Student : public Person {
    int s-id;
public:
    Student(/*...*/);
    Student(OC_APL*);
    ~Student(){destroy(OC_false)};
    OC_Type *getDirectType();
    void destroy(OC_Boolean aborted=OC_false);
};
```

General Form of Destroy() Implementation.

```
void  
This_Class::destroy(OC_Boolean aborted){  
    // class-specific cleanup code  
    if (aborted){  
        first_base_class::destroy(aborted);  
        second_base_class::destroy(aborted);  
    }  
}
```

Example:

```
void  
Student::destroy(OC_Boolean aborted){  
    if (aborted)  
        Person::destroy(aborted);  
}
```

Reference: Direct Reference (Handout 2-24)

```
class Person : public OC_Object {
    int age;
    Dog* mydog; // Direct Reference
public:
    Person(int its_age, char *its_name=NULL);
    Person(OC_API*);
    ~Person() {destroy(OC_false);}
    OC_Type *getDirectType();
    void destroy(OC_Boolean aborted=OC_false);
    void putObject(OC_Boolean
                  deallocate=OC_false);
    void deleteObject(OC_Boolean
                     deallocate=OC_true);
    Dog *dog();
    void dog(Dog* d);
};
```

How to use Direct Reference.

```
Person *p;
Dog      *d;

OC_transactionStart();
  p = new Person(20, "Mr.Right");
  d = new Dog("Doggie");
  d->putObject();
  p->dog(d);           // Question1
                    // Question3
OC_transactionCommit();

OC_transactionStart();
  p = (Person*)OC_lookup("Mr.Right");
  cout << (p->dog())->name(); // Question2
OC_transactionCommit();
```

Question1: How to use OC_directAssignObject().

```
void Person::dog(Dog* d) {  
    // instead of mydog = d; you have to  
    OC_directAssignObject((OC_Entity**) &mydog,  
        (OC_Entity**) &d);  
};
```

- This function must be used in place of regular pointer assignment.

Question2: How to use OC_directActivateObject().

```
dog *Person::dog() {  
    if (mydog)  
        if (mydog == OC_inactive)  
            OC_directActivateObject  
                ((OC_Entity**) &(mydog));  
    return mydog;  
};
```

Reference: Abstract Reference relieves the user of having to be aware of the storage location of its referent, and hides whether or not the referent is currently in memory. (textbook 24.5.2)

```
class Person : public OC_Object {
    int age;
    OC_Reference mydog;    // Abstract Reference
public:
    Person(int its_age, char *its_name=NULL);
    Person(OC_APL*);
    ~Person(){destroy(OC_false)};
    OC_Type *getDirectType();
    void destroy(OC_Boolean aborted=OC_false);
    void putObject(OC_Boolean deallocate=OC_false);
    void deleteObject(OC_Boolean
        deallocate=OC_true);
    Dog *dog();
    void dog(Dog* d);
};
```

How to initialize Abstract Reference.

```
Person::Person(/*...*/) {
    OC_Type *PersonType;

    PersonType = getDirectType();
    initDirectType(PersonType);

    mydog.initToNull();
    // ...
}
```

How to reference to another object through abstract reference.

```
void Person::dog(Dog* d) {
    // instead of mydog = d, you have
    mydog.reset(d, this);
    // ...
}
```

How to get the memory address of an object pointed to by abstract reference.

```
Dog* Person::dog() {  
    return (Dog*)mydog.binding(this);  
}
```

Question 3: If there are references pointed to other objects in a class, you may need to redefine `pubObject()`, `deleteObject()` and `destroy()` functions of the class.

- Handling of the deallocate argument of both `putObject()` and `deleteObject()` depends on how `destroy()` is written.

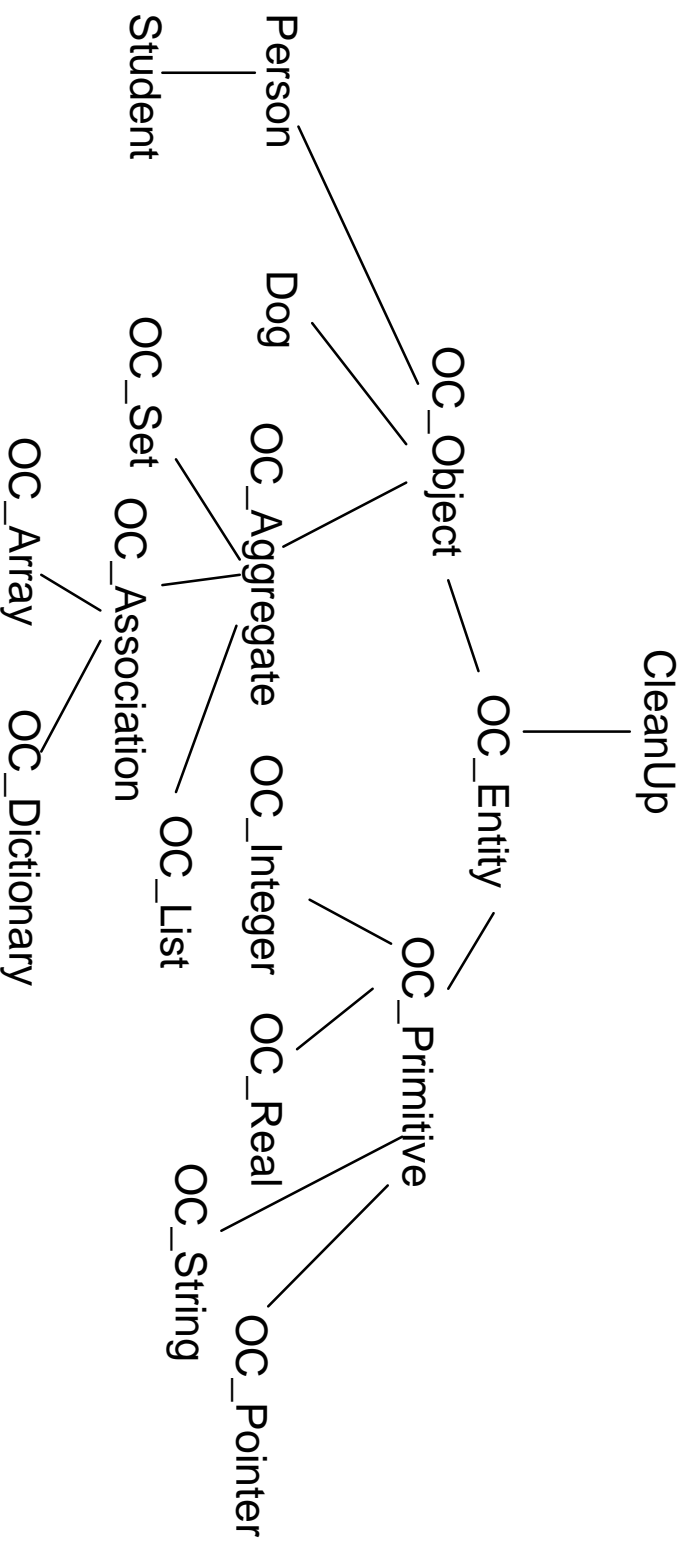
When to reimplement pubObject() and deleteObject()

```
void Person::destroy(OC_Boolean aborted){
    if (mydog) delete mydog;
    if (aborted)
        OC_Object::destroy(aborted);
}

void Person::putObject(OC_Boolean dealloc){
    if (mydog)
        mydog->putObject(OC_false);
    OC_Object::putObject(dealloc);
}

void Person::deleteObject(OC_Boolean dealloc)
    if (mydog)
        mydog->deleteObject(OC_false);
    OC_Object::deleteObject(dealloc);
}
```

Aggregates



Is Reference a persistent class?

A person may have many dogs.

```
class Person : public OC_Object {
    int age;
    OC_Reference mydogs; // a set of dogs
public:
    Person(int its_age, char *its_name=NULL);
    Person(OC_APL*);
    ~Person(){destroy(OC_false)};
    OC_Type *getDirectType();
    void destroy(OC_boolean aborted=OC_false);
    void putObject(OC_boolean deallocate=OC_false);
    void deleteObject(OC_boolean
        deallocate=OC_true);
    void remove_dog();
    void add_dog(Dog* d);
    void dump();
    // ...
};
```

How to initialize.

```
Person::Person( /* ... */ )
{
    OC_Type *PersonType;

    PersonType = getDirectType();
    initDirectType(PersonType);

    mydogs.initToNull();
    // ...
}
```

How to create a set object and insert a dog object into the set.

```
void Person::add_dog(Dog* d){
    OC_Type *DogType =
        (OC_Type*)OC_lookup("Dog");
    if (!mydogs.binding(this)){
        OC_Set *the_set = new(this,
            OC_defaultClustering)
            OC_Set(DogType, (char*)0);
        mydogs.reset(the_set, this);
    }
    OC_Set *the_set =
        (OC_Set*)mydogs.binding(this);
    if (!the_set->isMember((OC_Entity*)d))
        the_set->insert((OC_Entity*)d);
}
```

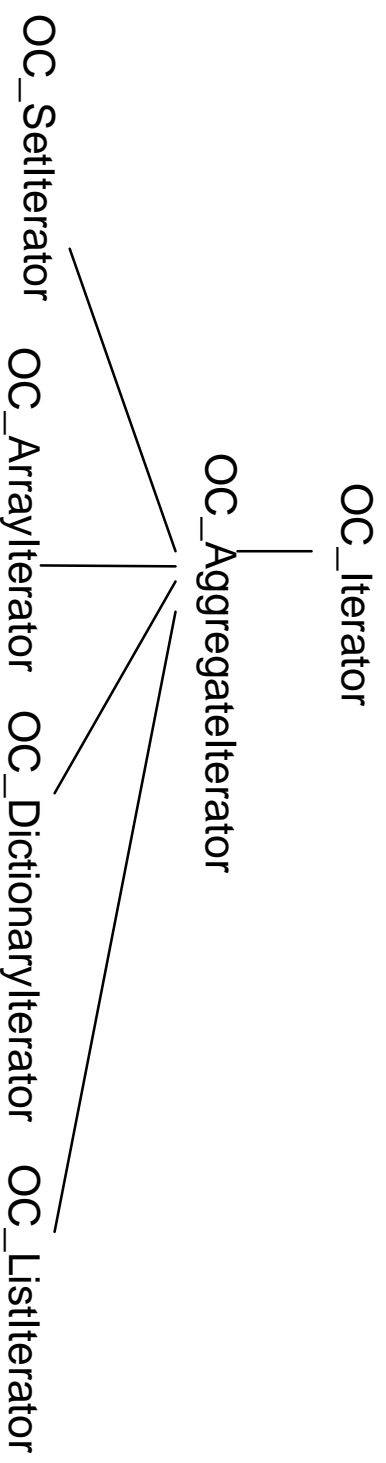
How to remove an object from a set object.

```
void Person::remove_dog(Dog* d){
    OC_Set *the_set =
        (OC_Set*)mydogs.binding(this);
    if (the_set->isMember((OC_Entity*)d)
        the_set->remove((OC_Entity*)d);
}
```

How to get all objects kept in a set object.

```
void Person::dump(){
    OC_Set *the_set = mydogs.binding(this);
    OC_SetIterator the_iterator(the_set);
    while (the_iterator.moreData()){
        ((Dog*)(OC_Entity*)the_iterator())->dump();
    }
}
```

Iterators: An important feature of aggregates is the ability to iterate through the elements of the aggregate.



destroy() Function

```
void Person::destroy(OC_Boolean aborted){
    Dog* d;
    if (mydogs.isActive(this)){
        Set* the_set = mydogs.binding(this);
        if (!the_set){
            OC_SetIterator the_iterator(the_set);

            while(the_iterator.moreData()){
                d = (Dog*)(Entity*)the_iterator();
                delete d;
            }
            delete the_set;
        }
    }
    if (aborted)
        OC_Object::destroy(aborted);}
}
```

putCluster() and deleteCluster(): two functions defined in the Object class.

```
void Person::putObject(OC_Boolean dealloc){
    OC_Set *the_set =
        (OC_Set*)mydogs.binding(this);
    if (!the_set)
        if(mydogs.isActive(this))
            the_set->putCluster(OC_false);
    OC_Object::putObject(dealloc);
}

void Person::deleteObject(OC_Boolean dealloc){
    OC_Set *the_set =
        (OC_Set*)mydogs.binding(this);
    if (!the_set)
        the_set->deleteCluster(OC_false);
    OC_Object::deleteObject(dealloc);
}
```