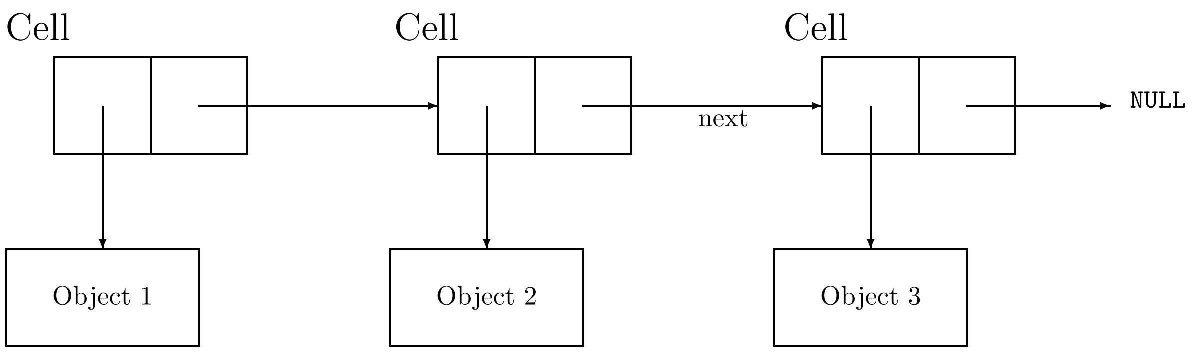
A Cell Class in C++



```
class Cell {
friend class Queue;
private:
   void *object;
   Cell *next;

public:
   Cell (void *obj, Cell *nxt) {
      object = obj;
      next = nxt;
   }
};
```

A Queue Class in C++

```
class Queue {
private:
   Cell *head, *tail;

public:
   Queue () { head = NULL;  tail = NULL; }

   void enqueue (void *obj) {
      if (head == NULL) {
         head = tail = new Cell(obj, NULL);
         return;
      }
      tail->next = new Cell(obj, NULL);
      tail = tail->next;
   }

   void *dequeue() {
      if (head == NULL) return NULL;
      void *obj = head->object;
      Cell *p = head;
      head = head->next;
      delete p;
      return obj;
   }

   bool empty () { return head == NULL; }
};
```

Using The Queue Class in C++

```
void main () {
   Queue *q = new Queue();
   q->enqueue(new int(10));
   q->enqueue(new int(11));
   q->enqueue(new int(12));
   cout << *(int*)q->dequeue() << " "
        << *(int*)q->dequeue() << " "
        << *(int*)q->dequeue() << "\n";
}
```

Adding Functions To The Class

```
class Queue {
private:
   Cell *head, *tail;
   void (*dispfn)(void *);

public:
   Queue (void (* d)(void *)) {
      head = NULL;  tail = NULL;  dispfn = d;
   }

   void enqueue (void *obj) {  ...  }
   void *dequeue() {  ...  }
   bool empty () { return head == NULL; }

   void display() {
      for (Cell *p = head ; p != NULL ; p = p->next)
         dispfn(p->object);
      cout << "\n";
   }
};

void intDisplay (void *obj) {  cout << *(int*)obj << " ";  }

void main () {
   Queue *q = new Queue(intDisplay);
   q->enqueue(new int(10));
   q->enqueue(new int(11));
   q->enqueue(new int(12));
   q->display();
}
```

Virtual Functions

```
class Object {
public:
   virtual void display() { cout << "Sorry\n";  }
};

class IntObject : public Object {
private:
   int number;

public:
   IntObject (int numb) { number = numb; }
   void display () {  cout << number << " ";  }
};

class StringObject : public Object {
private:
   char string[128];

public:
   StringObject (char *str) {  strcpy(string, str);  }
   void display () {  cout << string << " ";  }
};

class NullObject : public Object {
public:
   NullObject () {}
};
```

Make Sure All Stored Objects Have Function

```
class Cell {
friend class Queue;
private:
   Object *object;  // Only store Objects
   Cell *next;

public:
   Cell (Object *obj, Cell *nxt) { object = obj; next = nxt; }
};

class Queue {
public:
   Queue () {  head = NULL;  tail = NULL;  }
   void enqueue (Object *obj)  {  ...  }
   Object *dequeue()  {  ...  }
   bool empty () { ... }

   void display() {
      for (Cell *p = head ; p != NULL ; p = p->next)
         p->object->display();
      cout << "\n";
   }
};

void main () {
   Queue *q = new Queue();
   q->enqueue(new IntObject(10));
   q->enqueue(new StringObject("Hello---There"));
   q->enqueue(new NullObject());
   q->display();
}
```

From C++ to Java

```
C++:
     class Object {
     public:
        virtual void display() { cout << "Sorry\n";  }
     };

Java:
     class CellObject {    // Whoops!  Object is taken!
        void display() { System.out.println("Sorry"); }
     }

------------------------------------------------------------


C++:
     class IntObject : public Object {
     private:
        int number;

     public:
        IntObject (int numb) { number = numb; }
        void display () {  cout << number << " ";  }
     };

Java:
     class IntObject extends CellObject {
        int number;

        IntObject (int numb) { number = numb; }
        void display () {  System.out.print(number + " ");  }
     }
```

From C++ to Java

```
C++:
     class StringObject : public Object {
     private:
        char string[128];

     public:
        StringObject (char *str) {  strcpy(string, str);  }
        void display () {  cout << string << " ";  }
     };

Java:
     class StringObject extends CellObject {
        String string;

        StringObject (String str) {  string = str;  }
        void display () {  System.out.print(string + " ");  }
     }

-----------------------------------------------------------


C++:
     class NullObject : public Object {
     public:
        NullObject () {}
     };

Java:
     class NullObject extends CellObject {
        NullObject () {}
     }
```

From C++ to Java

C++:
```
     class Cell {
     private:
        Object *object;  // Only store CellObjects
        Cell *next;

     public:
        Cell (Object *obj, Cell *nxt) {
           object = obj;
           next = nxt;
        }
     };
```

Java:
```
     class Cell {
        CellObject object;
        Cell next;

        Cell (CellObject obj, Cell nxt) {
           object = obj;
           next = nxt;
        }
     }
```

From C++ to Java

C++:
```
     class Queue {
     private:
        Cell *head, *tail;

     public:
        Queue () {  head = NULL;  tail = NULL;  }
        void enqueue (Object *obj)  {  ...  }
        Object *dequeue()  {  ...  }
        bool empty () { return head == NULL; }

        void display() {
           for (Cell *p = head ; p != NULL ; p = p->next)
              p->object->display();
           cout << "\n";
        }
     };
```

Java:
```
     class Queue {
        Cell head, tail;

        Queue () {  head = null;  tail = null;  }
        void enqueue (CellObject obj) { ... }
        CellObject dequeue() { ... }
        boolean empty () { return head == null; }

        void display() {
           for (Cell p = head ; p != null ; p = p.next)
              p.object.display();
           System.out.println();
        }
     }
```

From C++ to Java

```
C++:
     void main () {
        Queue *q = new Queue();
        q->enqueue(new IntObject(10));
        q->enqueue(new StringObject("Hello---There"));
        q->enqueue(new NullObject());
        q->display();
     }

Java:
     public class Prog5 {
        public static void main (String argv[]) {
           Queue q = new Queue();
           q.enqueue(new IntObject(10));
           q.enqueue(new StringObject("Hello---There"));
           q.enqueue(new IntObject(12));
           q.enqueue(new NullObject());
           q.display();
        }
     }
```

An Employee Class

```
C++:
     class Employee {
     public:
        Employee () { name = NULL; }

        Employee (const char *nm) {
           this->name = new char[strlen(nm)+1];
           strcpy(this->name, nm);
        }

        char *getName () { return name; }
        virtual float computePay () const = 0;    // pure virtual
        virtual void display () {}
        virtual void setHours (float hrs) {}
        virtual void setSales (float sales) {}
        virtual void setSalary (float salary) { cout << "NO!\n"; }

     private:
        char *name;
     };

Java:
     class Employee {
        String name;

        Employee() { name = null; }
        Employee (String nm) { name = nm; } // overloaded '='
        String getName() { return name; }
        double computePay() { return 0.0; }
        double computeSalary() { return 0.0; }
        void display () {}
        void setHours(double hrs) {}
        void setSales(double sales) {}
        void setSalary(double salary) {}
     }
```

Calling Superclass Constructors

```
C++:
     class WageEmployee : public Employee {
     public:
        WageEmployee(const char *nm) : Employee(nm) {}
        WageEmployee(const char *nm, float r) : Employee(nm) {
           rate = r;
        }
        void setRate(float r)    { rate  = r; }
        void setHours(float hrs) { hours = hrs; }
        float getHours()         { return hours; }
        float getRate()          { return rate; }
        float computePay() const { return rate*hours; }
   }
     private:
        float rate;
        float hours;
     };

Java:
     class WageEmployee extends Employee {
        double rate;
        double hours;

        WageEmployee(String nm) {  super(nm); }
        WageEmployee(String nm, double r) {
           super(nm);
           rate = r;
        }
        void setRate(double r) { rate  = r; }
        void setHours(double hrs) { hours = hrs; }
        double getHours() { return hours; }
        double getRate() { return rate; }
        double computePay() { return rate*hours; }
     }
```

Functions Again

C++:

```
     class Queue {
     private:
        Cell *head, *tail;
        char *(*locator)(Object*);

     public:
        Queue (char *(*loc)(Object*)) {
          head = tail = NULL;
          locator = loc;
        }
        ...
        Object *find(char *id) {
           for (Cell *p = head ; p != NULL ; p = p->next)
              if (!strcmp(locator(p->object),id)) return p->object;
           return NULL;
        }

        void setLocator (char *(*f)(Object*)) { locator = f; }
     };

     char *nameFunc(CellObject *prog) {
        return ((Programmer*)prog)->getName();
     }

     char *identFunc(CellObject *prog) {
        return ((Programmer*)prog)->getIdent();
     }

     void main () {
        Queue *q = new Queue(identFunc);
        q->enqueue(new Programmer("Jim", "923-12-4422", 62.12));
        ...
        ((Employee*)(q->find("923-12-4422")))->setHours(0, 34);
        ...
        q->setLocator(nameFunc);
        ((Employee *)(q->find("Jim")))->display(0);
        ...
     }
```

Java Interface

```
Java:
     class Queue {
        Cell head, tail;
        FindFunc locator;   // FindFunc is an interface
        Queue (FindFunc loc) {
           head = tail = null;
           locator = loc;
        }
        ...
        Object_ find(String id) {
           for (Cell p = head ; p != null ; p = p.next)
              if (locator.find(p.object).equals(id)) return p.object;
           return null;
        }
        void setLocator (FindFunc loc) { locator = loc; }
     }

     interface FindFunc {  String find (Object_ obj);  }

     class NameFindFunc implements FindFunc {
        public String find (Object_ obj) {
           return ((Programmer)obj).getName();
        }
     }

     class IDFindFunc implements FindFunc {
        public String find (Object_ obj) {
           return ((Programmer)obj).getIdent();
        }
     }

     public static void main (String argv[]) {
        Queue q = new Queue(new IDFindFunc());
        q.enqueue(new Programmer("Jim", "923-12-4422", 62.12));
        ...
        ((Employee)(q.find("923-12-4422"))).setHours(0, 34);
        ...
        q.setLocator(new NameFindFunc());
        ((Employee)(q.find("Jim"))).display();
     }
```

Multiple Inheritance

```
class A {
   int a;

public:
   A() {}
   A(int x) { a=x; }
   int val() { return a; }
   int vA()  { return a; }
};

class B : public A {
   int a;

public:
   B() {}
   B(int x) { a=x; }
   int val() { return a; }
   int vB()  { return a; }
};

class C : public A {
   int a;

public:
   C(int x) { a=x; }
   int val() { return a; }
   int vC()  { return a; }
};

class D : public B, C {
   int a;

public:
   D(int x) : B(x+1), C(x+2) {  a = x; }
   int vD() { return a; }
   int vB() { return B::val(); }
   int vC() { return C::val(); }
};
```

```
A *a = new A(1);
B *b = new B(2);
C *c = new C(3);
D *d = new D(10);

                non-virtual   virtual
a->vA()             1            1
b->vB()             2            2
c->vC()             3            3
d->vD()             10           10

((B *)(d))->val()   11           11
d->vB()             11           11
((C *)(d))->val()   12           12
d->vC()             12           12

((A *)(a))->val()   1            1
((A *)(b))->val()   3            2
((A *)(c))->val()   5            3
((B *)(a))->val()   0            1
((B *)(b))->val()   2            2
((B *)(c))->val()   3            3
```

Multiple Inheritance

```
class A {
   int a;

   A() {}
   A(int x) { a=x; }
   int val() { return a; }
   int vA()  { return a; }
}

class B extends A {
   int a;

   B() {}
   B(int x) { a=x; }
   int val() { return a; }
   int vB()  { return a; }
}

class C {
   int a;

   C(int x) { a=x; }
   int val() { return a; }
   int vC()  { return a; }
}

class D extends B {
   C c;
   int a;

   D(int x) { super(x+1); c = new C(x+2); a = x; }
   int vD() { return a; }
   int vB() { return val(); }
   int vC() { return c.val(); }
}
```

```
A a = new A(1);
B b = new B(2);
C c = new C(3);
D d = new D(10);

a.vA()                1
b.vB()                2
c.vC()                3
d.vD()                10

((B)d).val()          11
d.vB()                11
((C)d).val()          --
d.vC()                12

((A)a).val()          1
((A)b).val()          2
((A)c).val()          --
((B)a).val()          --
((B)b).val()          2
((B)c).val()          --
```

Multiple Inheritance

C++:
```
     class SalesManager : public SalesPerson, public Manager {
     public:
        SalesManager(const char *nm, float w) :
           SalesPerson(nm, w), Manager(nm) { }

        // A must or else computePay() is ambiguous
        float computePay() const {
           return SalesPerson::computePay() + Manager::computePay();
        }

        void display() {
          SalesPerson::display();
          Manager::display();
        }
     };
```

Java:
```
     class SalesManager extends SalesPerson {
        Manager manager;

        SalesManager(String nm, double w) {
           super(nm, w);
           manager = new Manager(nm, w);
        }

        double computePay() {
           return super.computePay() + manager.computePay();
        }

        void display() {
           super.display();
           manager.display();
        }
     }
```

Reading From Files

```
C++:
      int cable_costs[100][100];
      char *buffer = new char[128];
      int city1, city2, cost;

      fstream fin("costs.dat", ios::in);
      while (fin.getline(buffer, 128, '\n')) {
         sscanf(buffer, "%d%d%d", &city1, &city2, &cost);
         cable_costs[city1][city2] = cost;
      }

Java:
      int cable_costs[][] = new int[100][100];
      int city1, city2, cost;
      BufferedReader is;
      String s;

      try {
         is = new BufferedReader(new FileReader("costs.dat"));
         while ((s = is.readLine()) != null) {
            try {
               StringTokenizer t = new StringTokenizer(s," ");
               city1 = Integer.parseInt(t.nextToken());
               city2 = Integer.parseInt(t.nextToken());
               cost  = Integer.parseInt(t.nextToken());
               cable_costs[city1][city2] = cost;
            }
            catch (NullPointerException e) { break; }
            catch (NoSuchElementException e) { break; }
         }
      }
      catch (IOException e) { }
```

Primitives Not Treated As Objects

```
C++:
     void main() {
         int *a = new int(1);           // OK
         int  b = *(int *)new int(2);   // OK
         int  c = new int(3);           // Not Allowed
         ...
     }

Java:
     class Int {
        int number;

        Int (int n) { number = n; }
        int value () { return number; }
     }

     public class Prog11 {
        public static void main (String argv[]) {
           Int a = new Int(1);          // OK
           Int b = new Int(new int(2)); // Not allowed
           int c = new int(3);          // Not allowed
           ...
        }
     }
```