DYNAMIC OBJECTS

DYNAMIC DATA MEMBERS

Dynamic data	accessible by pointer
	constructor initialization requires new to allocate memory
	destructor required to delete dynamic members

```
class Shape {
private:
      string name;
      int *size;                                               // pointer to heap memory
public:
      Shape(string n, int s): name(n), size( new int(n) ) { }   // allocate heap memory
      ~Shape() { delete size;}                                 // destructor: deallocate heap memory
}
```

DYNAMIC OBJECTS

Dynamic object   object memory stored on the heap and accessible by pointer

```
class Shape {
private:
      string name;
public:
      Shape(): name(“”) { }
};

Shape *s = new Shape();                    // allocate and initialize a dynamic object
delete s;                                  // deallocate a dynamic object
```

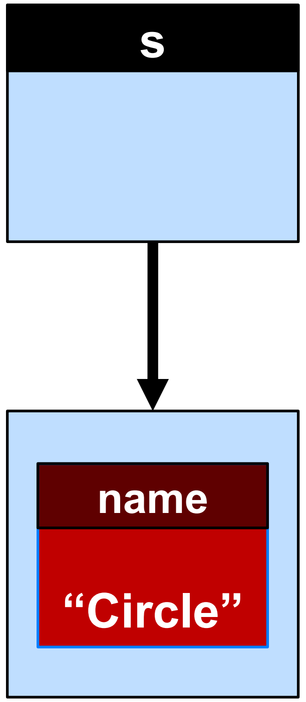
ARROW OPERATOR

```
class Shape {
private:
      string name;
public:
      Shape(): name(“”) {}
      Shape(string n): name(“”) {}
      string getName() const { return name; }
}

Shape *s = new Shape{“Circle”};

//deference the pointer s to access name
cout << (*s).getName();

// alternative syntax using arrow operator
cout << s->getName();
```



PASS POINTER OR REFERENCE

Concept Pass by reference reduces memory issues (memory leaks, dangling pointer, boundaries etc.

```
void print(Shape *s)                    // pass pointer by value
{
    std::cout << s->getSize() << "\n";
}
void print(Shape &s)                    // pass by reference (preferred in most cases)
{
    std::cout << s.getSize() << "\n";
}

Shape s1;
print(&s1);                             // syntax for automatic object to pass pointer by value
print(s1);                              // syntax for automatic object to pass by reference

Shape *s2 = new Shape{};
print(s2);                              // syntax for dynamic object to pass pointer by value
print(*s2);                             // syntax for dynamic object to pass by reference
```