Polymorphism Encyclopedia Article

Polymorphism

The following sections of this BookRags Literature Study Guide is offprint from Gale's For Students Series: Presenting Analysis, Context, and Criticism on Commonly Studied Works: Introduction, Author Biography, Plot Summary, Characters, Themes, Style, Historical Context, Critical Overview, Criticism and Critical Essays, Media Adaptations, Topics for Further Study, Compare & Contrast, What Do I Read Next?, For Further Study, and Sources.

The following sections, if they exist, are offprint from Beacham's Encyclopedia of Popular Fiction: "Social Concerns", "Thematic Overview", "Techniques", "Literary Precedents", "Key Questions", "Related Titles", "Adaptations", "Related Web Sites".

The following sections, if they exist, are offprint from Beacham's Guide to Literature for Young Adults: "About the Author", "Overview", "Setting", "Literary Qualities", "Social Sensitivity", "Topics for Discussion", "Ideas for Reports and Papers".

Contents

Polymorphism

The word "polymorphism" means "ability to take more than one form." Polymorphism in **object-oriented programming** specifically refers to the ability of a **programming** language to deal with objects differently, depending on their **data** type or **class**.

More important, it is the ability of the programming language to redefine or override methods in base classes with new methods in derived classes. This means that methods in derived classes have identical names, input parameters, and return values as methods in the **base class** but the program can figure out which one to **call** in any given situation, depending on the exact type of the **object**.

As an example, given a base class ThreeDimensionalObject, polymorphism enables the programmer to define different methods for calculating the volume of the shape for any number of classes derived from ThreeDimensionalObject. So, derived objects such as "Cube," "Sphere," "Cone," and "Dodecahedron" will all support the "CalculateVolume()" **method**, but polymorphism means that the **compiler** will ensure that the right version of the method is called for any given object.

In **C++** the classes could be declared like this:

- class ThreeDimensionalObject {
- public:
- virtual int CalculateVolume() const;
- };
-
- class Cube : public ThreeDimensionalObject {
- public:
- int CalculateVolume() const;
- };
-
- class Sphere : public ThreeDimensionalObject {
- public:
- int CalculateVolume() const;
- };

The "virtual" keyword tells the compiler it can override the **function** "CalculateVolume()" in any derived classes. If the programmer now creates a Cube and a Sphere, she will be able to get the volume by calling the CalculateVolume() method.

- Cube* myCube = new Cube();
- Sphere* mySphere = new Sphere();
- int cubeVol = myCubeCalculateVolume();
- int sphereVol = mySphereCalculateVolume();

However, it is also legal and possible to do this:

- ThreeDimensionalObject* myObjPtr = myCube;
- cubeVol = myCubeCalculateVolume();
- myObjPtr = mySphere;
- sphereVol = mySphereCalculateVolume();

And the compiler will still call the correct method. This is because the compiler chooses which method to call depending on the type of the underlying object and not the type of the pointer.

Polymorphism is a requirement of any real object-oriented programming language. The polymorphism described above is often called "parametric polymorphism" to distinguish it from a second kind of polymorphism called "overloading," which allows the compiler to treat identical symbols ("operator overloading"), and functions ("function overloading") differently depending on the context in which they are being used.