Kevin Lane

Ruby WebCrawler Part II

1. Inheritance in object-oriented programming describes how one class can extend, or be built off of another class. The class that is built off of is the superclass or base class, and the extending class is called a subclass or derived class. This is different from polymorphism in object-oriented programming. When a subclass overrides a superclass' method, the programming language makes a decision whether to call the subclass method or the superclass method. Suppose I have method f() in superclass A and I override that method in subclass B. If I create a new object of class B and call f() on that object, the programming language would call f() from class B due to the override. This is known as dynamic polymorphism. This is different from inheritance, but polymorphism cannot be discussed without inheritance. Polymorphism is implemented using the concepts of inheritance.

2. Polymorphism in imperative languages like C describes how you can create functions that have the same name with different parameters. This is also known as function overloading. This is different from polymorphism in an object-oriented programming language like Java, which is described above in question 1. Function overloading does not deal with inheritance in any way; in function overloading, the programming language determines which of the same named functions to call based off of the parameters given in the function call. In object-oriented polymorphism, the programming language determines which function to call based on what class the object was initialized with. If I created a new object by calling the class B constructor, then if I called f() (assuming we're in the same example as in question 1), it would be called from the B class. This is the case regardless of whether or not I created the object as a class A object or a class B object.

3. Encapsulation hides the representation of data by providing methods; it decouples the representation of data from the functionality of the data. In contrast, abstraction is generalization of structures. For example, C_++ has a standard template library with all data structures that are not type specific; the data structures are abstract, and they are assigned a type when a structure is called. Thus, abstraction hides process details, whereas encapsulation hides representation details.