

Application Programs vs. Software Components (reusable classes)

- The user of an *application program* is generally an *end user*, a person who's using the program to solve their problem, and whose main concern is that the program
 - does what it's supposed to do, and
 - behaves predictably *no matter what the input*.
- The user of a *reusable class* is not an end user but another programmer.
- We document application programs with *user manuals*; we document classes with a list of methods, including preconditions and postconditions for each method.
- `cs1.Keyboard` and `java.lang.Math` are software components.

Writing Your Own ADT Classes

- Sets of reusable classes are the basis for *all* modern software development, regardless of language or operating system.
- An Abstract Data Type (ADT) class provides the necessary support for a data structure: the ability to instantiate objects of the class, and a set of methods to manipulate the objects.
- An object has *state* (a set of data values, sometimes called *fields* or *attributes*) and *behavior* (a set of methods). The methods act on the data values, which changes the state.
- The *data values* are usually *private*, so that a calling program can't access them directly but must call methods.
- The methods are usually *public*, so a calling program can use them. Sometimes there are also *private methods*; why would we want these?

Characteristics of Data Values and Methods

- *Public* methods and data values are visible and available to any program that imports the class; *private* methods and data values are "invisible" outside the class itself and can be directly referred to only within the class.
- A *static* method or data value is associated with the *class*; a *non-static* one is associated with *each object*. That is, each object has its own "copy" of the method or data value.
- A *value-returning* method is one that returns a value to the calling program; a *void* method is one that does not return a value. At this time we'll consider both value-returning and void methods.

Common kinds of methods

- *Constructor*: creates and returns object from a class. If it has a list of parameters, it uses these to set the object's data values. The constructor's name is always that of the class; it has no return statement.
- *Modification or "Setter" Method*: A setter method can change the object's state. It usually has parameters, which it uses to modify one or more of an object's data values.
- *Accessor or "Getter"*: A getter method does not modify the state of the object, but just provides information about it. It usually has no parameters; it returns one of the object's data values, or maybe does a computation on these values.
- *input*: reads an objects data values from the keyboard or from a file.
- *toString*: collects (some of) an object's data values together into a string, suitable for displaying or writing to a file.

Testing a Software Component

- The main question you should ask yourself in starting to develop a test plan is "suppose I were going to spend a lot of money to buy this component for use in a critical system. What would I need to know about the component in order to be certain that the results I get from it are reliably correct? How can I have confidence that I am getting my money's worth?"
- In testing a component, we need to answer these questions:
- Does each method work properly by itself?
- Are the results of the methods *composable*? That is, can we always use the result of one method as an input to another method of the class?

- Putting this differently, we are testing to make sure that if the preconditions for each method hold before we call the method, then the postconditions hold after the method is called.
- For example, suppose you're testing an arithmetic operation for some new data type. Can we validly use the result of an addition as the input to a multiplication? Are you sure? Have you tested this?
- Another example: Suppose you're testing an output method that writes a structure to a file, and an input that reads the same kind of structure from a file. Can the read operation validly read a structure that the write operation wrote? Are you sure? Have you tested this?

Black-box (Behavioral) and White-box (Structural) Testing

- Black-box test design treats the system as a "black-box", so it doesn't explicitly use knowledge of the internal structure. Black-box test design is usually described as focusing on testing functional requirements. Synonyms for black-box include: behavioral, functional, opaque-box, and closed-box.
- White-box test design allows one to peek inside the "box", and it focuses specifically on using internal knowledge of the software to guide the selection of test data. Synonyms for white-box include: structural, glass-box and clear-box.
- In practice, it hasn't proven useful to use a single test design method. One has to use a mixture of different methods so that they aren't hindered by the limitations of a particular one. Some call this "gray-box" or "translucent-box" test design, but others wish we'd stop talking about boxes altogether.