CS1007 lecture #14 notes

thu 24 oct 2002

• news
  – exams will be back on tuesday
• wrapper classes
• inheritance
• this keyword
• reading: ch 7

classes.

• classes are the block around which Java is organized
• classes are composed of
  – data elements:
    * variables — i.e., their values can change during the execution of a program
    * constants — i.e., their values CANNOT change during the execution of a program
  – like variables, they have a type, a name and a value
• methods
  – modules that perform actions on the data elements
    * like variables, they have a type, a name and a value
  – unlike variables, the type can be void, which means that they don’t really have a value
  – constructors — special types of methods used to set up an object before it is used for the first time
• groups of related classes are organized into packages

the java.lang package.

• the superclass for all Java classes, at the top of the hierarchy
  – java.lang.Object
• wrappers around primitive data types; classes that define numeric limits and contain conversion methods
  – java.lang.Boolean
  – java.lang.Character
• string handling functions
  – java.lang.String
• also includes java.lang.Math

example wrapper class.

• java.lang.Integer class
• it is a wrapper around the int primitive data type
• it provides methods for converting between int and String
• a constructor:
  public Integer( int value );
• some constants:
  public static final int MIN_VALUE
  public static final int MAX_VALUE
• some methods:
  public int intValue();
  public static String toString( int i );
  public static Integer valueOf( String s );
• other wrapper classes are similar — see on-line documentation
java.lang.String class.

- this is a special wrapper class
- it wraps around char[]
- some constructors:
  - public String();
  - public String(String value);
- some methods:
  - public char[] toCharArray();
  - public static String valueOf(int i);
  - public int charAt(int index);
  - public int compareTo(String anotherString);
  - public int length();

inheritance.

- inheritance is the means by which classes are created out of other classes
- it is a cornerstone of object-oriented programming
- the idea is to create classes that can be re-used from one application to another
- classes contain data objects and methods
- you want to be able to change the data type of the data objects and still be able to use the same methods
- you also want to be able to change the flavor of what the methods do

inheritance tree (1).

- think of the most primitive Java class, Object as being at the root of the inheritance tree
- all other classes are "children" or subclasses of that class
- here is an example of the inheritance tree for Integer:
  ```
  Object
  ├── Number
  │    └── Integer
  │         ├── Integer is a subclass of Number and Number is a subclass of Object
  │         ├── Integer is also a subclass of Object
  │         └── conversely a parent is also called a superclass
  │              ├── Object is a superclass of Number and Number is a superclass of Integer
  │              ├── Object is also a superclass of Integer
  │              └── Object is also called the base class of Integer
  ```

inheritance tree (2).

- as you move DOWN the inheritance tree from the root to the leaf, you are extending subclasses from parent classes
  - parent classes are also called superclasses
  - or base classes
  - children classes are derived from their parents
- as you move UP the inheritance tree from the leaf to the root, you can say that each subclass is a more specific version of its parent
- this is known as the is-a relationship between a subclass and the parent class that the child extends
- the keyword this is used to specify a member of the current or immediate class
overriding methods.

- when you extend a class, you can override methods defined in the parent class by defining them again in the child (and giving the child version different behavior)
- the rule is: the version of any method that is invoked is the definition closest to the leaf of the tree
- if you want to refer to the version of the method in a class’s superclass, you use the super reference

overloading methods.

- in addition to changing precisely what a method does, you can also change the arguments to that method
- this is very useful if you are changing the data type of data objects defined in the class
- you can create a new version of a method which has different arguments from the version of the method defined in the class’s superclass
- this is what happens when we use different versions of the println() method:

```java
int i = 5;
String s = "hello";
System.out.println( i );
System.out.println( s );
```

other terminology...

- **polymorphism**
  - “having many forms”
  - lets us use different implementations of a single class
  - we talked about this in relation to interfaces
  - a polymorphic reference can refer to different types of objects at different times
- **abstract** class
  - represents a generic concept in a class hierarchy
  - cannot be instantiated — can only be extended