Abstract Data Type

- Counter is an example of an Abstract Data Type (ADT) – an abstraction representing a particular type of data
- Classes allow us to implement ADTs
  - The data and methods combine to implement the functionality we desire or expect for this type of data
  - The implementation details are hidden from the user
  - The type can be used in many different places in the program or in many programs

Package

- In general, each class is in a separate file
  - The name of the file should match the class name (with .java at the end)
- All classes in the same directory are part of the same package
- Whether or not a method is in the same class or package as the data or method it is accessing affects what it can see and do.

Data Hiding

- It is desirable to hide the inner details of a class (ADT) from the users of the class
- We want to be able to determine the correctness of our class without having to examine the entire program that it is used in
- For example, with our Counter class, we want to ensure that the value doesn’t change by more than 1.

Data Hiding

- Accessing instance variables from outside the class violates the data hiding principle

```java
class CounterTest2 { 
    public static void main( String[] args ) { 
        Counter c = new Counter();
        c.value = 150;
        System.out.println( "c.value = " + c.get() );
    }
}
```
Public/Private/Default
• private methods/fields cannot be accessed from outside of the class
• public methods/fields can be accessed from anywhere
• default (no specifier) methods/fields have package access – they can be accessed from other classes in the same package
  – if you don’t specify a package (see section 12.11), all classes in the same directory are part of the same default, un-named package

A Better Counter Class
• Use the private and public access specifiers to control access to instance variables and methods

class Counter {
    private int value;
    public void reset() { value = 0; }
    public void increment() { value++; }
    public void decrement() { value--; }
    public int get() { return value; }
}

Constructing Objects
• We create – or construct – objects with the new operator
  StringBuffer sb = new StringBuffer();
  Counter countThis = new Counter();
• This allocates memory for the object and initializes the object

Constructing Objects
• An object is initialized by calling its constructor
  – A constructor is a special method in a class
  – It has the same name as the class
  – No return type is specified
    • Implicit return type is instance of class
  – It is automatically called when an object of that type is created
  – Constructors are usually used to set data in the object to an initial value
  – Constructors can take parameters
Constructing Objects

• If the class definition does not include a constructor, then Java uses a default, no-argument constructor
  – This initializes all instance variables
    • booleans to false
    • other primitives to 0
    • everything else to null

Counter: Another Look

• This class does not have a defined constructor
• The default, no-argument constructor initializes the value field to 0
  – It is invoked when new Counter() is executed

class Counter {
  private int value;
  public void reset() { value = 0; }
  public void increment() { value++; }
  public void decrement() { value--; }
  public int get() { return value; }
}

Adding Constructors to Counter

• Want to create a Counter with an initial value

class Counter {
  private int value;
  public Counter() { value = 0; }
  public Counter( int v ) { value = v; }
  public void reset() { value = 0; }
  public void increment() { value++; }
  public void decrement() { value--; }
  public int get() { return value; }
}

Using Constructors

• Create two counters using our constructors

class CounterTest3 {
  public static void main( String[] args ) {
    Counter countThis = new Counter(10);
    Counter countThat = new Counter();
    countThis.increment();
    countThis.increment();
    countThat.increment();
    countThat.increment();
    System.out.println("countThis: " + countThis.get());
    System.out.println("countThat: " + countThat.get());
  }
}
Constructor Error

- Java provides the default, no-argument constructor only if there are no constructors defined in the class.
  - If we had only provided this constructor
    ```java
    public Counter( int v ) { value = v; }
    ```
  - Then creating a Counter with
    ```java
    Counter countThat = new Counter();
    ```
    would be a syntax error, because there is no constructor that takes 0 arguments.

Another Example

- Complex Numbers, with real and imaginary parts

  ```java
  public class Complex {
    private double real;
    private double imaginary;
    public Complex() { real = 0; imaginary = 0; }
    public Complex(double r, double i) {
      real = r;
      imaginary = i;
    }
    public double getReal() { return real; }
    public double getImaginary() { return imaginary; }
  }
  ```

Another Example

- Make some complex numbers

  ```java
  Complex a = new Complex();
  Complex b = new Complex( 3.7, 5.1 );
  Complex c = new Complex( 3, 0 );
  ```

A Dice Example

```java
import java.util.Random;

class Dice {
  private int die1, die2;
  private Random roller;
  Dice(int seed) { roller = new Random(seed); }
  void roll() {
    die1 = roller.nextInt(6) + 1;
    die2 = roller.nextInt(6) + 1;
  }
  int getTotal() { return die1 + die2; }
  public String toString() {
    return die1 + ", " + die2;
  }
}
```
Testing the Dice

```java
import tio.*;

class DiceTest {
    public static void main(String[] args) {
        System.out.println("Enter the seed.");
        Dice dice = new Dice(Console.in.readInt());
        System.out.println("How many times should I roll?");
        int count = Console.in.readInt();
        while(count > 0) {
            dice.roll();
            System.out.println("You rolled " + dice);
            System.out.println("The total is " + dice.getTotal());
            count--;
        }
    }
}
```

A Person Class

```java
class Person {
    private String name;
    private int age;
    private char gender;

    public Person( String n, int a, char g ) {
        name = n;
        age = a;
        gender = g;
    }

    public String getName() { return name; }
    public int getAge() { return age; }
    public char getGender() { return gender; }
}
```

Testing the Person Class

```java
class PersonTest {
    public static void main( String[] args ) {
        Person bill = new Person( "Bill", 25, 'M' );
        Person amy = new Person( "Amy", 33, 'F' );
        printPerson( bill );
        printPerson( amy );
    }

    static void printPerson( Person p ) {
        System.out.println( "Name: " + p.getName() );
        System.out.println( "Age: " + p.getAge() );
        System.out.println( "Gender: " + p.getGender() );
    }
}
```

toString()

- You can use `System.out.println()` to display any object
- However, the default behavior is not very useful
- You can write your own `toString()` method for your classes

```java
public String toString() {
    // Your custom implementation
}
```
- By providing every class with a `toString()` method, we can use `System.out.println()` to print ANY object value
Person.toString()

class Person {
    private String name;
    private int age;
    private char gender;

    public Person( String n, int a, char g ) {
        name = n;
        age = a;
        gender = g;
    }

    public String toString() {
        return "Name: " + name + 
                "Age: " + age + 
                "Gender: " + gender;
    }
}

Using toString()

class PersonTest2 {
    public static void main( String[] args ) {
        Person bill = new Person( "Bill", 25, 'M' );
        Person amy = new Person( "Amy", 33, 'F' );

        System.out.println( bill );
        System.out.println( amy );
    }
}

Static Methods and Variables

- Remember, a static (or class) method is one that is not associated with an object
- Static methods are invoked using the class name
  - Math.random()
  - String.valueOf()
- Similarly, a static or class variable is one that is independent of the objects of the class

Static Variables

- Class variables are associated with the class rather than with a particular instance of the class
- As with static methods, public static variables are accessed using the class name followed by the variable name
  - Math.PI
- A static variable is created once, when the class is first encountered by Java.
How Many Counters?

• Add a static variable to keep track of the number of Counter objects

```java
class Counter {
    private int value;
    private static int howMany = 0;

    public Counter() { value = 0; howMany++; }
    public Counter( int v ) { value = v; howMany++; }
    public void reset() { value = 0; }
    public void increment() { value++; }
    public void decrement() { value--; }
    public int get() { return value; }
    public static int howMany() { return howMany; }
}

class CounterTest4 {
    public static void main( String[] args ) {
        Counter countThis = new Counter(10);
        Counter countThat = new Counter();
        countThis.increment();
        countThat.increment();
        System.out.println("countThis: "+ countThis.get());
        System.out.println("countThat: "+ countThat.get());
        System.out.println("How Many: "+ Counter.howMany());
    }
}
```

Class Variables in Memory

```
<table>
<thead>
<tr>
<th>Class Variables</th>
<th>Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>Counter</td>
<td>Local variables in main()</td>
</tr>
<tr>
<td></td>
<td>CountThis CountThat Value 12 howMany 2</td>
</tr>
<tr>
<td></td>
<td>Class Counter howMany</td>
</tr>
</tbody>
</table>
```