Advanced Java Programming

After mastering the basics of Java you will now learn more complex but important programming concepts as implemented in Java.

James Tam

Commonly Implemented Methods

- The particular methods implemented for a class will vary depending upon the application.
- However two methods that are commonly implemented for many classes:
  - toString
  - equals

James Tam

"Method: toString"

- It's commonly written to allow easy determination of the state of a particular object (contents of important attributes).
- This method returns a string representation of the state of an object.
- It will automatically be called whenever a reference to an object is passed as a parameter is passed to the "print/println" method.
- Location of the online example:
  - /home/233/examples/advanced/toStringExample
  - www.cpsc.ucalgary.ca/~tamj/233/examples/advanced/toStringExample

James Tam

Class Person: Version 1

```
public class Person
{
    private String name;
    private int age;
    public Person () {name = "No name"; age = -1; }
    public void setName (String aName) { name = aName; }
    public String getName () { return name; }
    public void setAge (int anAge) { age = anAge; }
    public int getAge () { return age; }
}
```

James Tam

Class Person: Version 2

```
public class Person2
{
    private String name;
    private int age;
    public Person2 () {name = "No name"; age = -1; }
    public void setName (String aName) { name = aName; }
    public String getName () { return name; }
    public void setAge (int anAge) { age = anAge; }
    public int getAge () { return age; }

    public String toString ()
    {
        String temp = "";
        temp = temp + "Name: "+ name + "\n";
        temp = temp + "Age: " + age + "\n";
        return temp;
    }
}
```

James Tam

The Driver Class

```
class Driver
{
    public static void main (String args [])
    {
        Person p1 = new Person ();
        Person2 p2 = new Person2 ();
        System.out.println(p1);
        System.out.println(p2);
    }
}
```

James Tam

“Method: equals”

- It’s written in order to determine if two objects of the same class are in the same state (attributes have the same data values).
- Location of the online example:
  - /home/233/examples/advanced/equalsExample
  - www.cpsc.ucalgary.ca/~tamj/233/examples/advanced/equalsExample

James Tam

The Driver Class

```
public class Driver
{
    public static void main (String args [])
    {
        Person p1 = new Person ();
        Person p2 = new Person ();
        if (p1.equals(p2) == true)
            System.out.println ("Same");
        else
            System.out.println ("Different");

        p1.setName ("Foo");
        if (p1.equals(p2) == true)
            System.out.println ("Same");
        else
            System.out.println ("Different");
    }
}
```

James Tam

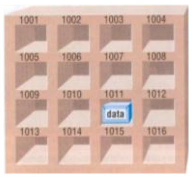
The Person Class

```
public class Person
{
    private String name;
    private int age;
    public Person () {name = "No name"; age = -1; }
    public void setName (String aName) { name = aName; }
    public String getName () { return name; }
    public void setAge (int anAge) { age = anAge; }
    public int getAge () { return age; }
    public boolean equals (Person aPerson)
    {
        boolean flag;
        if ((name.equals(aPerson.getName())) && (age == aPerson.getAge ()))
            flag = true;
        else
            flag = false;
        return flag;
    }
}
```

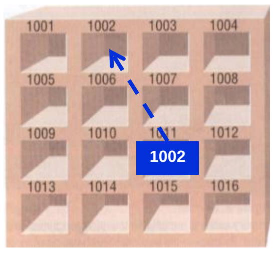
James Tam

Reminder: Variables

•Think of the ‘mail box’ metaphor



•A slot containing a reference (and not data) will contain the number of another ‘slot’ (address) in memory.



Pictures from Computers in your future by Pfaffenberger B

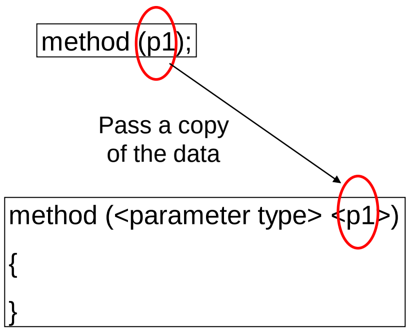
James Tam

Methods Of Parameter Passing

- Passing parameters as value parameters (pass by value)
- Passing parameters as variable parameters (pass by reference)

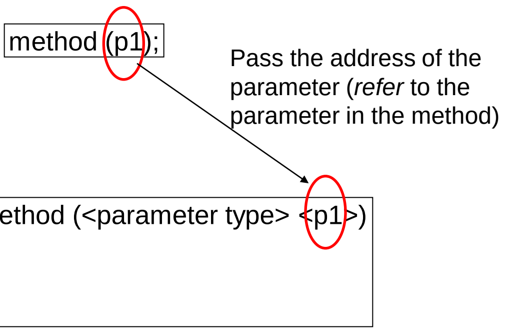
James Tam

Passing Parameters As Value Parameters



James Tam

Passing Parameters As Reference Parameters


Pass the address of the parameter (*refer* to the parameter in the method)

```
method (p1);
```

```
method (<parameter type> <p1>)
{
}
```

Parameter Passing In Java: Simple Types

•All simple types are always passed by value in Java.

Type	Description
byte	8 bit signed integer
short	16 but signed integer
int	32 bit signed integer
long	64 bit signed integer
float	32 bit signed real number
double	64 bit signed real number
char	16 bit Unicode character
boolean	1 bit true or false value

Parameter Passing In Java: Simple Types (2)

- Location of the online example:
  - /home/233/examples/advanced/valueParameters
  - www.cpsc.ucalgary.ca/~tamj/233/examples/advanced/valueParameters

```
 public static void main (String [] args)
{
       int num1;
       int num2;
       Swapper s = new Swapper ();
       num1 = 1;
       num2 = 2;
       System.out.println("num1=" + num1 + "\tnum2=" + num2);
       s.swap(num1, num2);
       System.out.println("num1=" + num1 + "\tnum2=" + num2);
}
```

James Tam

Passing Simple Types In Java (2)

```
public class Swapper
{
    public void swap (int num1, int num2)
    {
        int temp;
        temp = num1;
        num1 = num2;
        num2 = temp;
        System.out.println("num1=" + num1 + "\tnum2=" + num2);
    }
}
```

James Tam

Passing References In Java

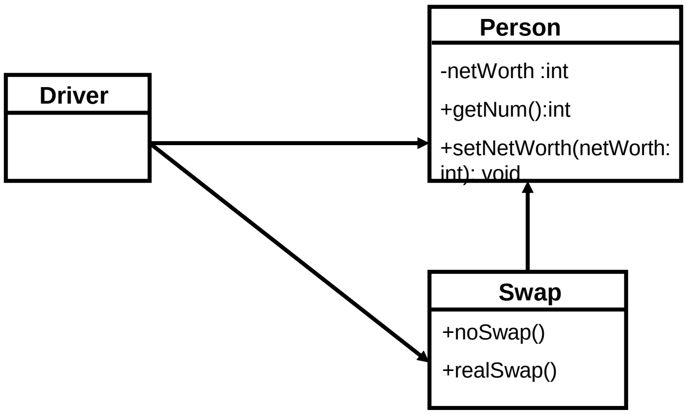
- (Reminder: References are required for variables that are arrays or objects)
- Question:
  - If a reference (object or array) is passed as a parameter to a method do changes made in the method continue on after the method is finished?

  Hint: If a reference is passed as a parameter into a method then a copy of the reference is what is being manipulated in the method.

James Tam

An Example Of Passing References In Java: UML Diagram

- Location of the online example:
  - /home/233/examples/advanced/referenceParameters
  - www.cpsc.ucalgary.ca/~tamj/233/examples/advanced/referenceParameters



James Tam

An Example Of Passing References In Java: The Driver Class

```
public static void main(String [] args) {
    Person jamesTam, billGates;
    Swap s1;
    jamesTam = new Person();
    billGates = new Person();
    s1 = new Swap ();
    jamesTam.setNetWorth(1);
    billGates.setNetWorth(2000000000)
    System.out.println("Before swap:\t tam=" + jamesTam.getNetWorth()
        + "\tgates=" +       billGates.getNetWorth());
    s1.noSwap(jamesTam,billGates);
    System.out.println("Before swap:\t tam=" + jamesTam.getNetWorth() +
        "\tgates=" +       billGates.getNetWorth());
    s1.realSwap(jamesTam,billGates);
    System.out.println("Before swap:\t tam=" + jamesTam.getNetWorth() +
                    "\tgates=" +       billGates.getNetWorth());
  }
```

James Tam

An Example Of Passing References In Java: Class Person

```
public class Person {
   private int netWorth;

   public Person() {
          netWorth = 0;
   }

   public int getNetWorth() {
          return netWorth;
   }

   public void setNetWorth(int newWorth) {
          netWorth = newWorth;
   }
}
```

James Tam

An Example Of Passing References In Java: Class Swap

```
public class Swap
{
    public void noSwap (Person p1, Person p2)
    {
        Person temp;
        temp = p1;
        p1 = p2;
        p2 = temp;
        System.out.println("In noSwap\t p1=" + p1.getNetWorth () + "\tp2=" +
                                      p2.getNetWorth());
    }
```

James Tam

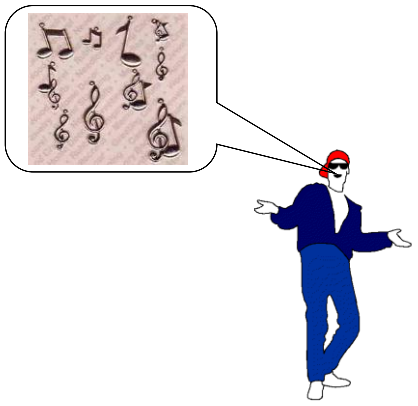
An Example Of Passing References In Java: Class Swap (2)

```
    public void realSwap (Person p1, Person p2)
    {
        Person temp = new Person();
        temp.setNetWorth(p1.getNetWorth());
        p1.setNetWorth(p2.getNetWorth());
        p2.setNetWorth(temp.getNetWorth());
        System.out.println("In noSwap\t p1=" + p1.getNetWorth () + "\tp2=" +
                                      p2.getNetWorth());
    }
}  // End of class Swap
```

James Tam

Passing By Reference For Simple Types

- It cannot be done directly in Java
- You must use a wrapper!



James Tam

Wrapper Class

- A class definition built around a simple type

e.g.,

```
public class IntegerWrapper
{
   private int num;
   public int getNum () { return num; }
   public void setNum (int newNum) { num = newNum; }
}
```

- Wrapper classes are also used to provide class-like capabilities to simple variable types e.g., class Integer

James Tam

References: Things To Keep In Mind/Reminder

- If you refer to just the name of the reference then you are dealing with the reference (to an object, to an array).
  - E.g., f1 = f2;
  - This copies an address from one reference into another reference, the original objects don't change.

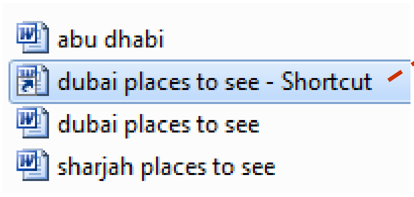
- If you use the dot-operator then you are dealing with the actual object.
  - E.g.,
  - temp = f2;
  - temp.setNum (f1.getNum());
  - temp and f2 refer to the same object and using the dot operator changes the object which is referred to by both references.
- Other times this may be an issue
  - Assignment
  - Comparisons

James Tam

Shallow Copy Vs. Deep Copies

- Shallow copy (new term, concept should be review)
  - Copy the address from one reference into another reference
  - Both references point to the same dynamically allocated memory location
  - e.g.,

```
Foo f1;
Foo f2;
f1 = new Foo ();
f2 = new Foo ();
f1 = f2;
```



A shortcut ('link' in UNIX) is similar to a shallow copy. Multiple things that refer to the same item (document)

James Tam

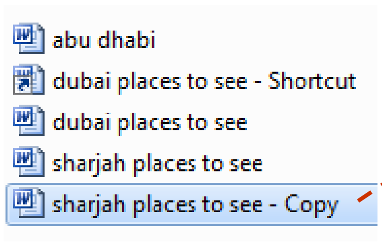
Shallow Vs. Deep Copies (2)

- Deep copy (new term, concept should be review)
  - Copy the contents of the memory location referred to by the reference
  - The references still point to separate locations in memory.
  - e.g.,

```
f1 = new Foo ();
f2 = new Foo ();
f1.setNum(1);
f2.setNum(f1.getNum());
System.out.println("f1=" + f1.getNum() + "\tf2=" + f2.getNum());
f1.setNum(10);
f2.setNum(20);
System.out.println("f1=" + f1.getNum() + "\tf2=" + f2.getNum());
```

James Tam

Shallow Vs. Deep Copies (3)



Making an actual physical copy is similar to a deep copy.

James Tam

Comparison Of References Vs. Data(Objects)

- Location of the online example:
  - /home/233/examples/advanced/comparisionsReferencesVsObjects
  - www.cpsc.ucalgary.ca/~tamj/233/examples/advanced/comparisionsReferencesVsObjects

```
public class Person
{
   private int age;
   public Person () { age = -1; }
   public void setAge (int anAge) { age = anAge; }
   public int getAge () { return age; }
}
```

James Tam

Comparison Of The References

```
public class DriverReferences
{
   public static void main (String [] args)
   {
      Person p1 = new Person ();
      Person p2 = new Person ();
      p1.setAge(1);
      p2.setAge(p1.getAge());
      if (p1 == p2)
         System.out.println("References: Same location");
      else
         System.out.println("References: different locations");

   }
}
```

James Tam

Comparison Of The Data

```
public class DriverData
{
   public static void main (String [] args)
   {
       Person p1 = new Person ();
       Person p2 = new Person ();
       p1.setAge(1);
       p2.setAge(p1.getAge());
       if (p1.getAge() == p2.getAge())
           System.out.println("Data: Same information");
       else
           System.out.println("Data: different information");
   }
}
```

Implementing an 'equals' method once allows the method to be used many times when equality checks must occur

James Tam

A Previous Example Revisited: Class Sheep

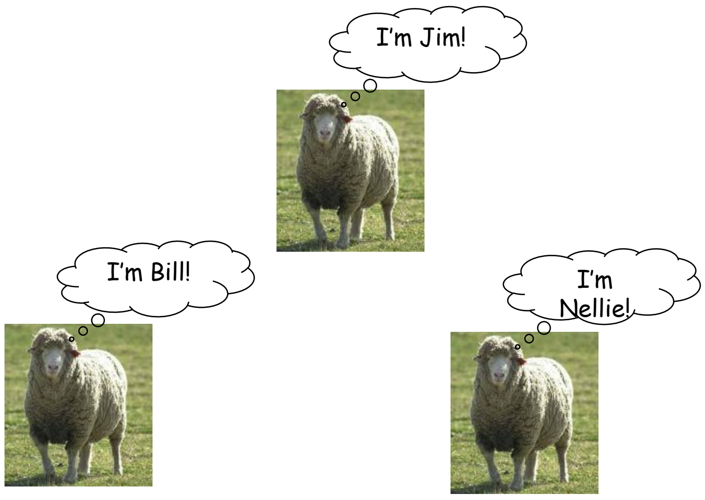
```
public class Sheep
{
   private String name;

   public Sheep ()
   {
      System.out.println("Creating \"No name\" sheep");
      name = "No name";
   }
   public Sheep (String aName)
   {
      System.out.println("Creating the sheep called " + n);
      setName(aName);
   }
   public String getName () { return name;}

   public void setName (String newName) { name = newName; }
}
```

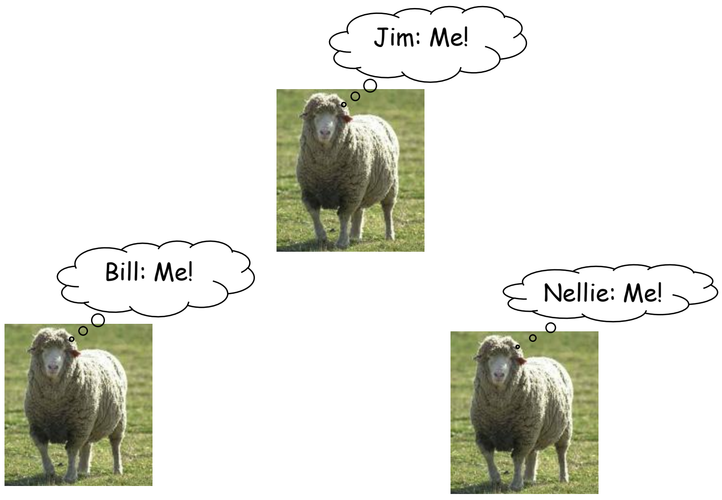
James Tam

We Now Have Several Sheep



James Tam

Question: Who Tracks The Size Of The Herd?

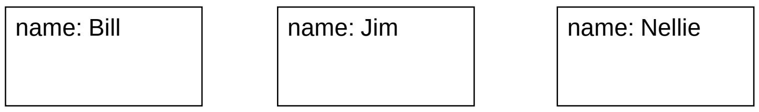


James Tam

Answer: None Of The Above!

- Information about all instances of a class should not be tracked by an individual object.
- So far we have used instance fields.
- Each *instance* of an object contains *it's own set of instance fields* which can contain information unique to the instance.

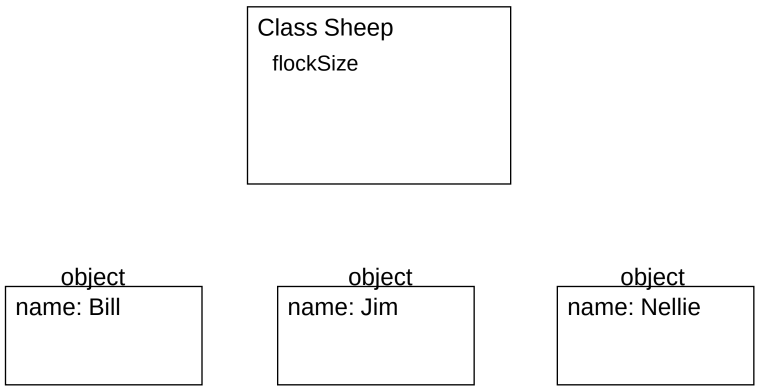
```
public class Sheep
{
        private String name;
       :          :         :
}
```



James Tam

The Need For Static (Class Fields)

- Static fields: One instance of the field exists *for the class* (not for the instances of the class)



James Tam

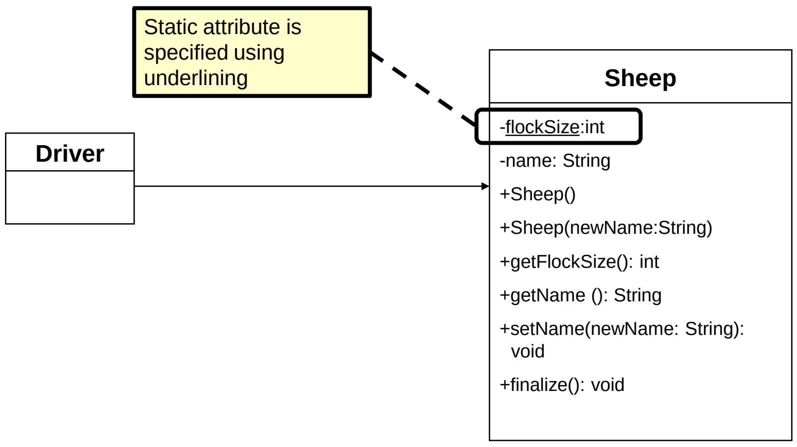
Static (Class) Methods

- Are associated with the class as a whole and not individual instances of the class.
- Typically implemented for classes that are never instantiated e.g., class Math.
- May also be used act on the class fields.

James Tam

Static Data And Methods: UML Diagram

- Location of the online example:
  - /home/233/examples/advanced/staticExample
  - www.cpsc.ucalgary.ca/~tamj/233/examples/advanced/staticExample



James Tam

Static Data And Methods: The Driver Class

```
public class Driver
{
    public static void main (String [] args)
    {
        System.out.println();
        System.out.println("You start out with " + Sheep.getFlockSize() + "
        sheep");
        System.out.println("Creating flock...");
        Sheep nellie = new Sheep ("Nellie");
        Sheep bill = new Sheep("Bill");
        Sheep jim = new Sheep();
```

James Tam

Static Data And Methods: The Driver Class (2)

```
        System.out.print("You now have " + Sheep.getFlockSize() + " sheep:");
        jim.setName("Jim");
        System.out.print("\t"+ nellie.getName());
        System.out.print(", "+ bill.getName());
        System.out.println(", "+ jim.getName());
        System.out.println();
    }
}  // End of Driver class
```

James Tam

Static Data And Methods: The Sheep Class

```
public class Sheep
{
  private static int flockSize = 0;
  private String name;

   public Sheep ()
    {
       flockSize++;
       System.out.println("Creating \"No name\" sheep");
       name = "No name";
    }

    public Sheep (String aName)
    {
       flockSize++;
       System.out.println("Creating the sheep called " + newName);
       setName(aName);
    }
```

James Tam

Static Data And Methods: The Sheep Class (2)

```
    public static int getFlockSize () { return flockSize; }

    public String getName () {  return name; }

    public void setName (String newName)  { name = newName; }

    public void finalize ()
    {
       System.out.print("Automatic garbage collector about to be called for ");
       System.out.println(this.name);
       flockSize--;
    }
}// End of definition for class Sheep
```

James Tam

Accessing Static Methods/Attributes

- Inside the class definition

Format:

<*attribute or method name*>

Example:

```
public Sheep ()
 {
     flockSize++;
 }
```

James Tam

Accessing Static Methods/Attributes (2)

- Outside the class definition

Format:

<*Class name*>.<*attribute or method name*>

Example:

```
Sheep.getFlockSize();
```

James Tam

Rules Of Thumb: Instance Vs. Class Fields

- If a attribute field can differ between instances of a class:
  - The field probably should be an instance field (non-static)
- If the attribute field relates to the class (rather to a particular instance) or to all instances of the class
  - The field probably should be a static field of the class

James Tam

Rule Of Thumb: Instance Vs. Class Methods

- If a method should be invoked regardless of the number of instances that exist (e.g.., the method can be run when there are no instances) then it probably should be a static method.
- If it never makes sense to instantiate an instance of a class then the method should probably be a static method.
- Otherwise the method should likely be an instance method.

James Tam

Static Vs. Final

- **Static**: Means there's one instance of the field for the class (not individual instances of the field for each instance of the class)
- **Final**: Means that the field cannot change (it is a constant)

```
public class Foo
{
    public static final int num1= 1;
    private static int num2;            /*  Rare */
    public final int num3 = 1;          /* Why bother? */
    private int num4;
       :         :
}
```

James Tam

An Example Class With A Static Implementation

```
public class Math
{
  // Public constants
  public static final double E = 2.71…
  public static final double PI = 3.14…

  // Public methods
  public static int abs (int a);
  public static long abs (long a);
            :                 :
}
```

- For more information about this class go to:
  - http://java.sun.com/j2se/1.5.0/docs/api/java/lang/Math.html

James Tam

Should A Class Be Entirely Static?

- Generally it should be avoided if possible because it often bypasses many of the benefits of the Object-Oriented approach.
- Usually purely static classes (cannot be instantiated) have only methods and no data (maybe some constants).
- When in doubt *DO NOT* make attributes and methods static.
- Example candidate static class:

```
class Swapper
{
    public noSwap (… ) { }
    public realSwap (…) {}
}
```

The class has methods but no variable attributes (why are separate instances needed?)

James Tam

A Common Error With Static Methods

•Recall: The "this" reference is an implicit parameter that is automatically passed into the method calls (you've seen so far).

•e.g.,

•Foo f = new Foo ();

•f.setNum(10);

Explicit parameter

Implicit parameter "this"

James Tam

A Common Error With Static Methods

•Static methods have no “this” reference as an implicit parameter (because they are not associated with any instances).

```
public class Driver
{
   private int num;
   public static void main (String [] args)
   {
      num = 10;
   }
}
```

Compilation error:

```
Driver3.java:6: non-
static variable num
cannot be referenced
from a static context
      num = 10;
      ^
error
```

James Tam

Immutable Objects

•Once instantiated they cannot change (all or nothing)

```
e.g., String s = "hello";
      s = s + " there";
```

•Changes to immutable objects should be minimized

James Tam

Minimize Modifying Immutable Objects (2)

- If you must make many changes consider substituting immutable objects with mutable ones

e.g.,

```
public class StringBuffer
{
   public StringBuffer (String str);
   public StringBuffer append (String str);
      :        :            :        :

}
```

For more information about this class

- http://java.sun.com/j2se/1.5.0/docs/api/java/lang/StringBuffer.html

James Tam

3. Minimize Modifying Immutable Objects (3)

```
public class StringExample
{
   public static void main (String []
  args)
   {
       String s = "0";
       for (int i = 1; i < 100000; i++)
           s = s + i;
   }
}
```

```
public class StringBufferExample
{
   public static void main (String [] args)
    {
        StringBuffer s = new   StringBuffer("0");
        for (int i = 1; i < 100000; i++)
            s = s.append(i);
     }
}
```

James Tam

Be Cautious When Writing Accessor And Mutator Methods: First Version

•Location of the online example:
- /home/233/examples/advanced/securityVersion1
- www.cpsc.ucalgary.ca/~tamj/233/examples/advanced/securityVersion1

```
public class Driver
{
    public static void main (String [] args)
    {
        CreditInfo newAccount = new CreditInfo (10, "James Tam");
              newAccount.setRating(0);
        System.out.println(newAccount);
    }
}
```

James Tam

5. Be Cautious When Writing Accessor And Mutator Methods: First Version (2)

```
public class CreditInfo
{
    public static final int MIN = 0;
    public static final int MAX = 10;
    private int rating;
    private StringBuffer name;
    public CreditInfo ()
    {
        rating = 5;
        name = new StringBuffer("No name");
    }
    public CreditInfo (int newRating, String newName)
    {
        rating = newRating;
        name = new StringBuffer(newName);
    }

    public int getRating () { return rating;}
```

James Tam

5. Be Cautious When Writing Accessor And Mutator Methods: First Version (3)

```
public void setRating (int newRating)
{
    if ((newRating >=  MIN) && (newRating <= MAX))
        rating = newRating;
}

public StringBuffer getName ()
{
    return name;
}

public void setName (String newName)
{
    name = new StringBuffer(newName);
}
```

James Tam

5. Be Cautious When Writing Accessor And Mutator Methods: First Version (4)

```
    public String toString ()
    {
        String s = new String ();
        s = s + "Name: ";
        if (name != null)
        {
            s = s + name.toString();
        }
        s = s + "\n";
        s = s + "Credit rating: " + rating + "\n";
        return s;
    }
} // End of class CreditInfo
```

James Tam

Be Cautious When Writing Accessor And Mutator Methods: Second Version

(All mutator methods now have private access).

- Location of the online example:
  - /home/233/examples/advanced/securityVersion2
  - www.cpsc.ucalgary.ca/~tamj/233/examples/advanced/securityVersion2

James Tam

Be Cautious When Writing Accessor And Mutator Methods: Second Version (2)

```
public class Driver
{
    public static void main (String [] args)
    {
        CreditInfo newAccount = new CreditInfo (10, "James Tam");

        StringBuffer badGuyName;
        badGuyName = newAccount.getName();

            badGuyName.delete(0, badGuyName.length());
        badGuyName.append("Bad guy on the Internet");

        System.out.println(newAccount);
    }
}
```

James Tam

5. Be Cautious When Writing Accessor And Mutator Methods: Second Version (3)

```
public class CreditInfo
{
    private int rating;
    private StringBuffer name;

    public CreditInfo ()
    {
        rating = 5;
        name = new StringBuffer("No name");
    }

    public CreditInfo (int newRating, String newName)
    {
        rating = newRating;
        name = new StringBuffer(newName);
    }
```

James Tam

5. Be Cautious When Writing Accessor And Mutator Methods: Second Version (4)

```
     public int getRating ()
    {
        return rating;
    }
     private void setRating (int newRating)
    {
        if ((newRating >= 0) && (newRating <= 10))
            rating = newRating;
    }
    public StringBuffer getName ()
    {
        return name;
    }
     private void setName (String newName)
    {
        name = new StringBuffer(newName);
    }
```

James Tam

5. Be Cautious When Writing Accessor And Mutator Methods: Second Version (5)

```
    public String toString ()
    {
        String s = new String ();
        s = s + "Name: ";
        if (name != null)
        {
            s = s + name.toString();
        }
        s = s + "\n";
        s = s + "Credit rating: " + rating + "\n";
        return s;
    }
}
```

James Tam

5. Be Cautious When Writing Accessor And Mutator Methods: Third Version

- Location of the online example:
  - /home/233/examples/advanced/securityVersion3
  - www.cpsc.ucalgary.ca/~tamj/233/examples/advanced/securityVersion3

```
public class Driver
{
    public static void main (String [] args){
        CreditInfo newAccount = new CreditInfo (10, "James Tam");
        String badGuyName;
        badGuyName = newAccount.getName();

       badGuyName = badGuyName.replaceAll("James Tam", "Bad guy on
                                    the Internet");
              System.out.println(badGuyName + "\n");
         System.out.println(newAccount);
    }
}
```

James Tam

5. Be Cautious When Writing Accessor And Mutator Methods: Third Version (2)

```
public class CreditInfo
{
    private int rating;
    private String name;
    public CreditInfo ()
    {
        rating = 5;
        name = "No name";
    }
    public CreditInfo (int newRating, String newName)
    {
        rating = newRating;
        name = newName;
    }
    public int getRating ()
    {
        return rating;
    }
```

James Tam

5. Be Cautious When Writing Accessor And Mutator Methods: Third Version (3)

```
    private void setRating (int newRating)
    {
        if ((newRating >= 0) && (newRating <= 10))
            rating = newRating;
    }

    public String getName ()
    {
        return name;
    }

    private void setName (String newName)
    {
        name = newName;
    }
```

James Tam

5. Be Cautious When Writing Accessor And Mutator Methods: Third Version (4)

```
    public String toString ()
   {
       String s = new String ();
       s = s + "Name: ";
       if (name != null)
       {
           s = s + name;
       }
       s = s + "\n";
       s = s + "Credit rating: " + rating + "\n";
       return s;
   }
}
```

James Tam

5. Be Cautious When Writing Accessor And Mutator Methods

- When choosing a type for an attribute it comes down to tradeoffs, what are the advantages and disadvantages of using a particular type.
- In the previous examples:
  - Using mutable types (e.g., StringBuffer) provides a speed advantage.
  - Using immutable types (e.g., String) provides additional security

James Tam

Important Terminology, Concepts, Methods

- toString()
- equals()
- pass by value
- pass by reference
- Wrapper classes
- Deep/shallow copy
- Static/class attributes/methods
- Instance attributes
- Static vs. final keywords
- Immutable vs. mutable

James Tam

After This Section You Should Now Know

- Two useful methods that should be implemented for almost every class: toString and equals
- What is the difference between pass by value vs. pass by reference
- The difference between references and objects
- Issues associated with assignment and comparison of objects vs. references
- The difference between a deep vs. a shallow copy
- What is a static method and attribute, when is appropriate for something to be static and when is it inappropriate (bad style)
- What is the difference between a mutable and an immutable type

James Tam

After This Section You Should Now Know (2)

- When should a mutable vs. immutable type be used and the advantages from using each type

James Tam