UNIVERSITÉ Concordia UNIVERSITY

CENTRE FOR CONTINUING EDUCATION

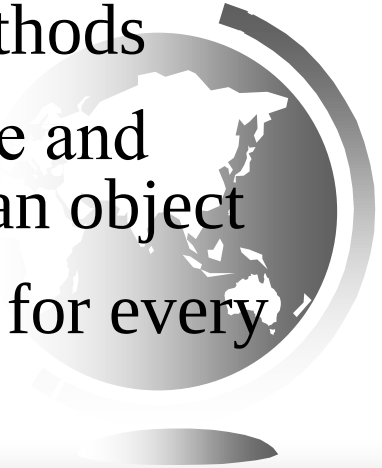
DESKTOP APPLICATION DEVELOPMENT WITH JAVA – CEJV569

Lecture #2

The Object

Class and Objects

- Object Oriented Programming organizes source code in units we call a class.
- Class is a description or blueprint for some unit of work or action we wish to perform in a program
- Methods in a class use the memory of a program to store the executable code of the methods
- Regardless of the number of objects that will be created there will only ever be one set of methods
- When we need to use this unit of work's code and variables we instantiate the class and create an object
- The variables in an object consume memory for every instance of an object created



Data and Actions

- Classes can be categorized by their purpose or goal
- Most all classes consist of data and actions
- What the actions, in the form of methods, do is how we define the category that class belongs
- Classes can be:
  - Data classes
  - Action classes
  - Hybrid classes combining data and actions

The Data Class

- Every problem requires that information must be acted upon
- Rarely is the information represented by a single value
- A set of data is the most common situation
- For example:
  - *You have been assigned the task of writing a program that will calculate the amount of paint needed to cover every wall in the house. Rooms can be different colours and in an individual room you can have different colours. Different brands of paint for different colours may be used.*

Data Class Design

- Data classes contain a set of values
- The set is defined by the aspect of the program domain that actions will be performed on.

Data Class Design – Java Bean

- Java Bean was originally a specification for designing Java classes for use by GUI builder tools.
- Parts of this specification have been adopted by a wide range of frameworks and libraries.
- We will use the core definition of a Java Bean for our coding of data classes
- We will use some of the core definition when coding action classes

Java Bean – Access Control

- Access level modifiers determine whether other classes can use a particular field or invoke a particular method.
- There are two levels of access control.
- At the top level—public, or package-private (no explicit modifier).
- A Java Bean should always have **public** top level access

Java Bean

```
public class Automoble {

}
```

Java Bean – Access Control

- At the member level—public, private, protected, or package-private (no explicit modifier).

Access Levels				
Modifier	Class	Package	Subclass	World
public	Y	Y	Y	Y
protected	Y	Y	Y	N
no modifier	Y	Y	N	N
private	Y	N	N	N

- The most restrictive is **private** and all bean member variable must be private

Java Bean

```
public class Automoble {
    private String manufacturer;
    private String model;
    private int
engineDisplacement;

}
```

Java Bean - Constructor

- A Java bean must have a default constructor
- The constructor may be empty in which case the member variables take on their default values
- Remember that only class variables have default values
- Method or local variables do not have default values
- If the default values are not appropriate then the default constructor can initialize the member variables

Java Bean

```
public class Automoble {
    private String manufacturer;
    private String model;
    private int engineDisplacement;

    public Automobile() {};
      OR
    public Automobile() {
        this.manufacturer = "";
        this.Model = "";
        this.engineDisplacement = 100;
}
```

Java Bean – setters and getters

- With access to the class variables prohibited methods must be used for access
- The actions that can be performed on a member value in a data bean is defined as:
  - Accessors
    - Read the current state of a member variable
  - Mutators
    - Change or replace the current state of a member variable
- These methods must be named following the convention for such methods

Java Bean – setters and getters

- Accessor methods must begin with the word get followed by a capital letter
- Accessor methods for boolean member variables may begin with the word "*is*" followed by a capital letter
- Accessor methods have no parameters and return a value
- Mutator methods must begin with the word set followed by a capital letter
- Mutator methods must have at least one parameter and return nothing (void)

Java Bean

```
public String getManufacturer() {
   return this.manufacturer;
}
public void setManufacturer(String manufacturer) {
   this.manufacturer = manufacturer;
}
public String getModel() {
   return this.model;
}
public void setModel(String model) {
   this.model = model;
}
public String getEngineDisplacement() {
   return this.engineDisplacement;
}
public void setEngineDisplacement(String engineDisplacement) {
   this.engineDisplacement = engineDisplacement;
}
```

Exercise 7

- The Stock class.
- Make sure to write the class as a Java Bean with all the mentioned principles.

Exercise 8

- The Account class.
- Make sure to write the class as a Java Bean with all the mentioned principles.

Object – The Ultimate Superclass

- Every class in Java extends the Object superclass except Object itself
- If a class does not explicitly extend a class then the compiler assumes it extends Object
- Defines the basic state and behavior that all objects should have
- Allows all objects to passed around without knowing the actual type
  - Expect an Object and anything can be delivered

The Object Interface

- public Object()
- public final Class getClass()
- protected void finalize() throws Throwable
- public int hashCode()
- public boolean equals(Object obj)
- public String toString()
- protected Object clone() throws
- CloneNotSupportedException
- public final void notify()
- public final void notifyAll()
- public final void wait(long timeout) throws
- InterruptedException
- public final void wait(long timeout, int nanoseconds) throws
- InterruptedException
- public final void wait() throws InterruptedException

public final Class getClass()

- All classes can return an object of type Class
- Class class has more than 50 methods
- Can be used to determine if two objects are the same class
  - Subclasses and Superclasses will not be equal
  - Use `instanceOf` if super and sub must compare

protected void finalize() throws Throwable

- Called by the garbage collector on an object when garbage collection determines that there are no more references to the object
- Garbage collector is not guaranteed to run at any specific time
- May never be called during the lifetime of a program
- Mistakenly compared to the Destructor function in C++
- If you rely on finalize for the correct operation of your application, then you're doing something wrong

notify and wait

- Used in concurrent programming
- We will examine these when we discuss threading

public int hashCode()

- Hash Code is an integer generated by a formula that uses the current state of the object
- It behaves somewhat like a primary key for an object
- Two objects of the same class and of the same state will have the same hash code
- Two objects of the same class and of different states may not have different hash codes
- Data structures that order objects by hash code perform faster than ordering based on objects such as the String
- Must be overridden in a subclass

public boolean equals(Object obj)

- All objects have both *identity* (the object's location in memory) and *state* (the object's data)
- == operator always compares identity
- `equals()` compares state
- `equals()` and `hashCode()` are always overridden together
- You must use the same set of fields in both of these methods.
- You are not required to use all fields.
- Must be overridden in a subclass

How to override the equals method of the Object class

The equals method of the Product class

```
@Override
public boolean equals(Object object)
{
    if (object instanceof Product)
    {
        Product product2 = (Product) object;
        if
        (
            code.equals(product2.getCode()) &&
            description.equals(
                product2.getDescription()) &&
            price == product2.getPrice()
        )
            return true;
    }
    return false;
}
```

How to override the equals method of the Object class (cont.)

The equals method of the LineItem class

```
@Override
public boolean equals(Object object)
{
    if (object instanceof LineItem)
    {
        LineItem li = (LineItem) object;
        if
        (
            product.equals(li.getProduct()) &&
            quantity == li.getQuantity()
        )
            return true;
    }
    return false;
}
```

public String toString()

- Intended to return a String representation of an object
- Commonly used for debugging

protected Object clone() throws CloneNotSupportedException

- There are two ways to copy an object
- Shallow Copy
  - Just copy the reference so that two variables point to the same memory location
- Deep Copy
  - Create a new object and then copy the state from the original to the copy
- `clone()` is intended to create a deep copy of itself

What to Override?

- There are three methods that every data bean class should override
- They are:
  - **`hashCode`**
  - **`equals`**
  - **`toString`**
- Write toString and equals methods for Stock and Account.

Exercise 9

- The Fan class.
- Make sure to write the class as a Java Bean with all the mentioned principles.
- Don’t forget to override toString and equals method.

Immutable Objects and Classes

If the contents of an object cannot be changed once the object is created, the object is called an *immutable object* and its class is called an *immutable class*. If you delete the set method in the Circle, the class would be immutable because radius is private and cannot be changed without a set method.

A class with all private data fields and without mutators is not necessarily immutable. For example, the following class Student has all private data fields and no mutators, but it is mutable.

Example

```
public class Student {
  private int id;
  private BirthDate birthDate;

  public Student(int ssn,
      int year, int month, int day) {
    id = ssn;
    birthDate = new BirthDate(year, month, day);
  }

  public int getId() {
    return id;
  }

  public BirthDate getBirthDate() {
    return birthDate;
  }
}
```

```
public class BirthDate {
  private int year;
  private int month;
  private int day;

  public BirthDate(int newYear,
      int newMonth, int newDay) {
    year = newYear;
    month = newMonth;
    day = newDay;
  }

  public void setYear(int newYear) {
    year = newYear;
  }
}
```

```
public class Test {
  public static void main(String[] args) {
    Student student = new Student(111223333, 1970, 5, 3);
    BirthDate date = student.getBirthDate();
    date.setYear(2010); // Now the student birth year is changed!
  }
}
```

What Class is Immutable?

For a class to be immutable, it must mark all data fields private and provide no mutator methods and no accessor methods that would return a reference to a mutable data field object.

Strings Are Immutable

A String object is immutable; its contents cannot be changed. Does the following code change the contents of the string?

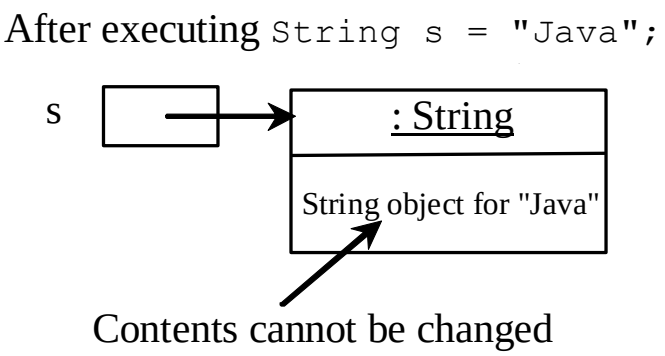
```
String s = "Java";
s = "HTML";
```

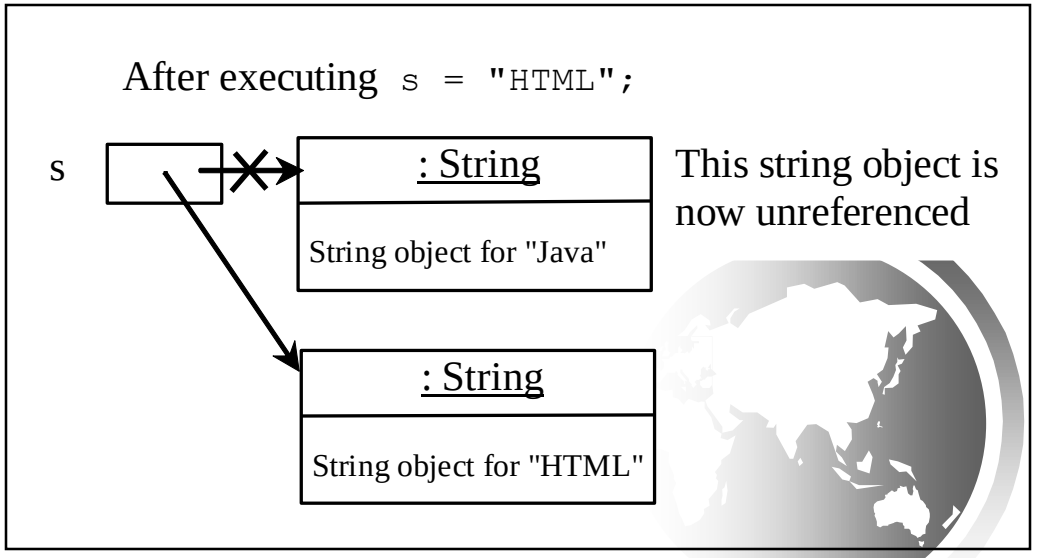
animation

Trace Code

```
String s = "Java";
```

```
s = "HTML";
```

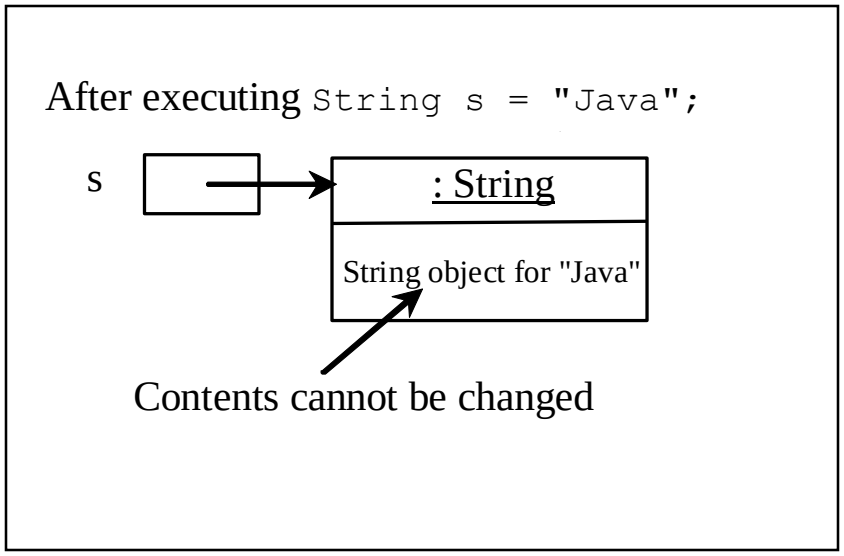


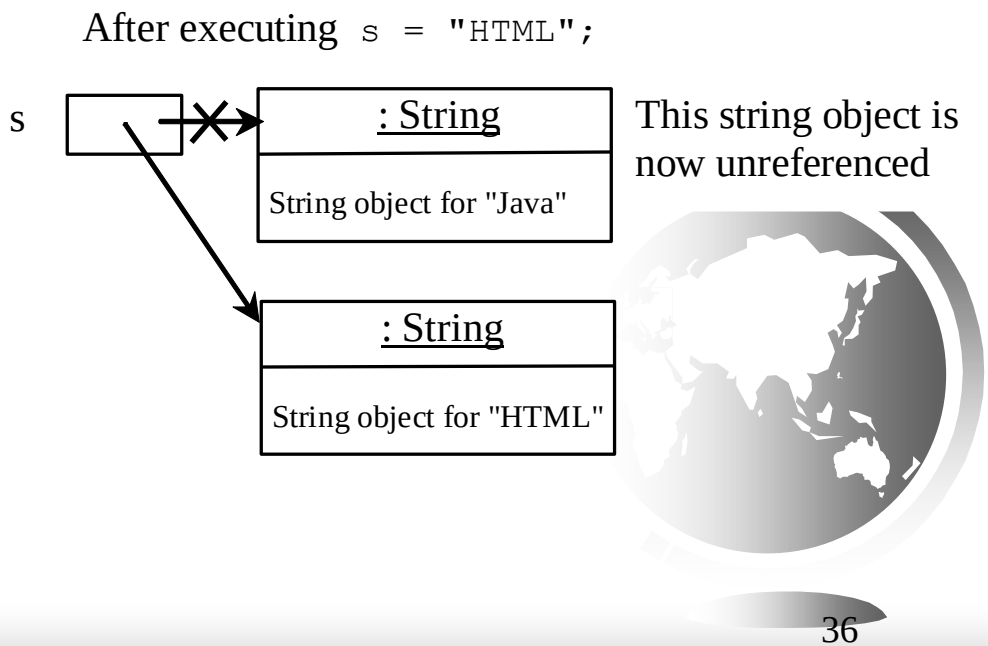


animation

Trace Code

```
String s = "Java";
s = "HTML";
```





Interned Strings

Since strings are immutable and are frequently used, to improve efficiency and save memory, the JVM uses a unique instance for string literals with the same character sequence. Such an instance is called *interned*. For example, the following statements:

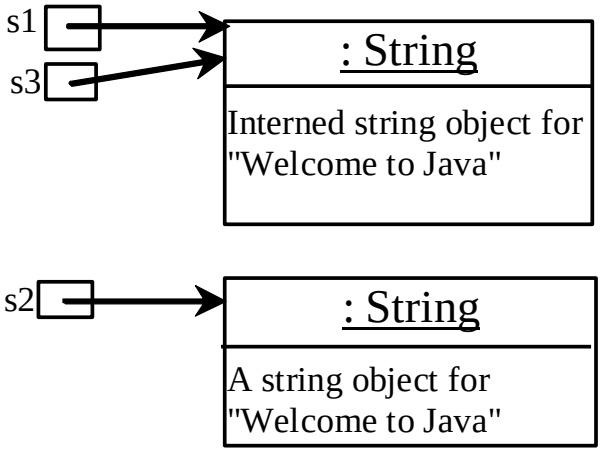
Examples

```
String s1 = "Welcome to Java";

String s2 = new String("Welcome to Java");

String s3 = "Welcome to Java";

System.out.println("s1 == s2 is " + (s1 == s2));
System.out.println("s1 == s3 is " + (s1 == s3));
```



display

s1 == s2 is false

s1 == s3 is true

A new object is created if you use the new operator.

If you use the string initializer, no new object is created if the interned object is already created.

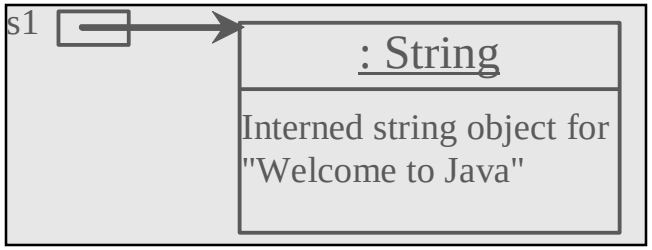
animation

Trace Code

```
String s1 = "Welcome to Java";

String s2 = new String("Welcome to Java");

String s3 = "Welcome to Java";
```

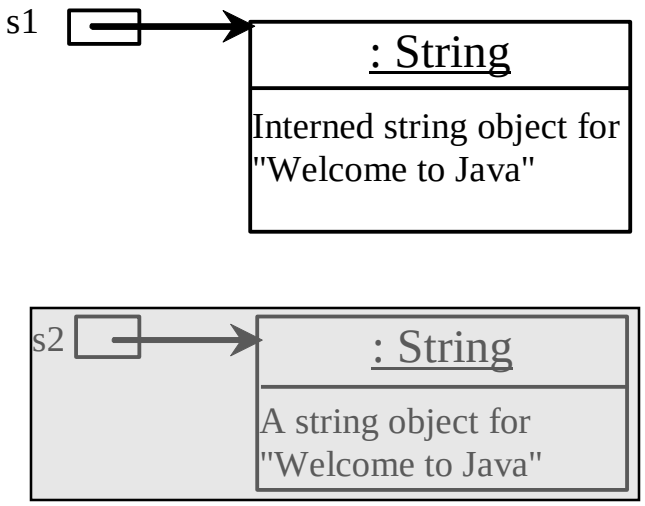


Trace Code

```
String s1 = "Welcome to Java";

String s2 = new String("Welcome to Java");

String s3 = "Welcome to Java";
```

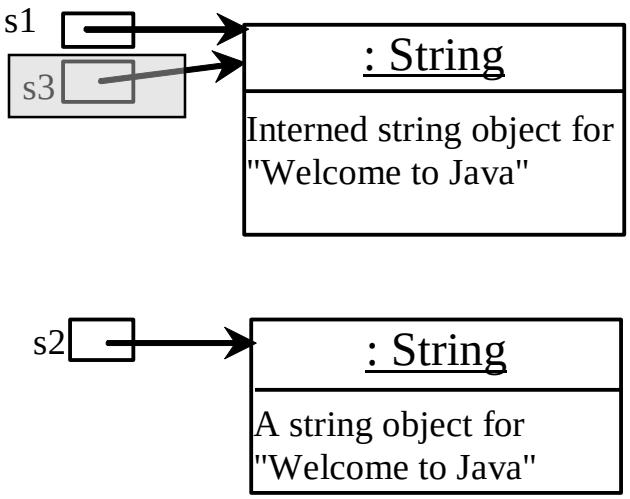


Trace Code

```
String s1 = "Welcome to Java";

String s2 = new String("Welcome to Java");

String s3 = "Welcome to Java";
```



Scope of Variables

- The scope of instance and static variables is the entire class. They can be declared anywhere inside a class.
- The scope of a local variable starts from its declaration and continues to the end of the block that contains the variable. A local variable must be initialized explicitly before it can be used.

The this Keyword

- The this keyword is the name of a reference that refers to an object itself. One common use of the this keyword is reference a class's *hidden data fields*.
- Another common use of the this keyword to enable a constructor to invoke another constructor of the same class.

Reference the Hidden Data Fields

```
public class F {
  private int i = 5;
  private static double k = 0;

  void setI(int i) {
    this.i = i;
  }

  static void setK(double k) {
    F.k = k;
  }
}
```

```
Suppose that f1 and f2 are two objects of F.
F f1 = new F(); F f2 = new F();

Invoking f1.setI(10) is to execute
   this.i = 10, where this refers f1

Invoking f2.setI(45) is to execute
   this.i = 45, where this refers f2
```

Calling Overloaded Constructor

```
public class Circle {
  private double radius;

  public Circle(double radius) {
    this.radius = radius;
  }

  public Circle() {
    this(1.0);
  }

  public double getArea() {
    return this.radius * this.radius * Math.PI;
  }
}
```

this must be explicitly used to reference the data field radius of the object being constructed

this is used to invoke another constructor

Every instance variable belongs to an instance represented by this, which is normally omitted

The JavaBean Rules

- Public class
- Private class variables
- Default constructor
- Mutators must begin with the word `set`
- Accessors must begin with the word `get` or `is` for booleans
- Class must be `serializable`
  - Serializable means that the class state can be stored in a file on disk
  - We will look at this when we examine reading and writing files
  - This can be ignored for now

Action Bean

- A class that performs actions on data
- Actions can be but not limited to:
  - Input
    - User input of data
    - Retrieving from a file
    - Retrieving data from a database
  - Processing
  - Output
    - Displaying data on a screen or printer
    - Storing data in a file
    - Storing data in a database

Action Bean

- Actions can be further classified by the category of the tasks such as:
  - Presentation
  - Database
  - Business
  - Network
- Tasks can then be classified by the problem they are solving
  - Salary calculation
  - Image processing

Action Bean

- Action beans use data beans
- This means that the action bean must have a reference to the data bean
- Usually class variables
- These data beans may be
  - created by the action bean, usually in the constructor
  - created by another class and passed in to the constructor or a set method

Interface

- A class interface is the public methods of a class that are accessible from the outside
- An interface class is a file that defines the methods that must be public in a regular class
- An interface class only consists of the return type and signature of a public method
- A regular class that implements an interface class must have methods that match what is described in the interface
- Java 8 introduced default methods that are implemented by having the method's code in the interface class

Action Class and Interface

- Action classes should implement an interface
- Over time the implementation of an algorithm may be improved
- To ensure that any new implementation can be used in an existing application it must have the same interface
- An interface class acts as a contract to ensure that new implementations are compatible with existing code.

Exercise 10

- MyPoint class
- Do not forget to write a test program that creates the two points (0,0) and (10,30.5) and displays the distance between them.

Exercise 11

- Making a bankroll
- You need to create 2 packages: first for Data bean and second for your application
- Make sure to create your interface first, and then implement the beans.