cisc3120
design and implementation of software applications I
spring 2015
lecture # I.6

topics:

- introduction to java, part 6
  - abstract classes
  - interfaces
  - member accessibility
  - UML basics
  - review of OOP concepts

abstract classes

- abstract classes allows to exploit the common features of classes
- it represents a generic concept in a class hierarchy
- cannot be instantiated – can only be inherited
- declared with the keyword `abstract` before `class`

```
public abstract class Vehicle {
   ...
}
```

- an abstract method is a method declared without implementation (without braces, and followed by a semicolon)

```
public abstract void move(double x, double y);
```

- an abstract class may or may not have abstract methods
- however a class with an abstract method have to be declared abstract.

abstract classes (2).

- when an abstract class is inherited, the subclass has to provide implementations for all of the abstract methods in its parent class.
- otherwise the derived class must also be declared abstract

```
public abstract class Vehicle {
   public abstract void move(double x, double y);
   public double getSpeed() { return speed; }
   private double speed ;
}
...
public class Car extends Vehicle {
   public void move(double x, double y) {
      // has to implement this method
   }
}
```

- abstract classes may define methods and declare data fields
- subclasses extending abstract classes inherit the defined methods

interfaces.

- an interface is *not a class* but a set of requirements for classes that want to conform to the interface
- methods cannot be implemented in the interface
- interfaces cannot have instance fields
- all methods in the interface are automatically public
- interfaces are used for:
  - design
  - interoperability
- do not confuse this use of the word interface with the same word in the graphical user interface (GUI)

interfaces (2).

- interfaces are declared using a similar syntax as classes
- to declare an interface you have to use keyword `interface` instead of `class`

```
public interface Movable {
   boolean move(double x, double y);
}
```

- like classes they can be extended

```
public interface Flyable extends Movable {
   void changeAltitude(int rate);
}
```

- like in abstract classes, you cannot instantiate an interface using `new`
- however you can declare interface variables
- interface variable must refer to an object that implements the interface

interfaces (3).

- any class that implements `java.lang.Comparable` interface is required to implement the `compareTo` method which takes an `Object` parameter and return and integer

```
public interface Comparable {
   int compareTo(Object other);
}
```

- in order to use an interface the classes must use the keyword `implements` in the class declaration

```
public class Car extends Vehicle implements Comparable {
  public int compareTo(Object other){
    Car c = (Car) other;
    if(this.vehicleIdentNum == c.getIdentNum()) { return 0; }
    else if(this.vehicleIdentNum < c.getIdentNum()) { return -1; }
    else { return 1; }
  }
}
```


interfaces(4).

- why bother with interfaces when we can declare classes abstract?

```
public abstract class Comparable {    // WHY NOT?
   public abstract int compareTo(Object other);
}
```

- Java doesn't allow multiple interitance, a class can only have a single parent

```
public class Car extends Vehicle, Comparable      // ERROR
```

- classes may implement from multiple interfaces

```
public class Car extends Vehicle implements Comparable, Movable
// OK
```

- interfaces provide most of the benefits of multiple inheritance while avoiding its complexities and inefficiencies

interfaces - design.

- interfaces are very useful for emphasizing functionality of classes during the initial design stages of large programs.

```
public interface Drivable {
   int getSpeed();
   void setSpeed(double speed);
   void steer();
   boolean move();
}
```

- interface describes a class, but does not say how the methods and the data fields are implemented
- it describes only the services provided by the class
- it represents the outward appearance of a class seen by the users of the class
- interfaces can also be used to describe an inheritance structure

interfaces - interoperability.

- example : a power cord with a plug
  - the design of the plug is standard
  - the design ensures that an appliance can be used anywhere
  - the adoption of a common design (interface) ensures interoperability
- Java interfaces can be used in similar fashion to ensure that objects exhibit common behavior
- example: `java.util.Arrays` class provides methods to perform operations on the data such as searching, sorting, etc.

```
public static void sort(Object[] a)
```

- in order to use the `sort` method on the object arrays, the class of the object has to implement `Comparable` interface.

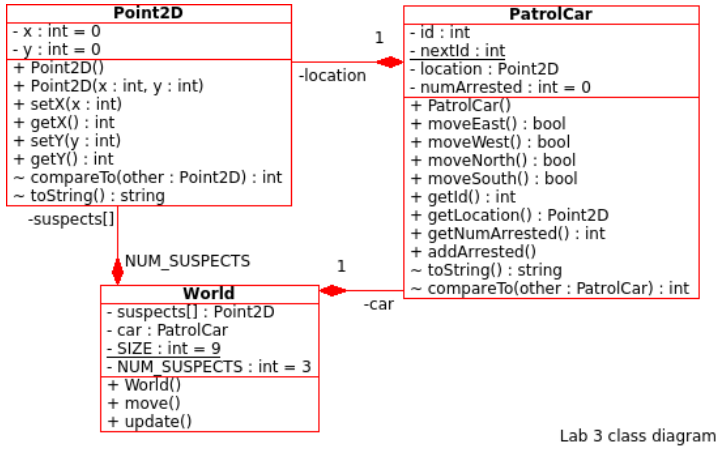
member accessibility.

- *private*: only accessible within the class they are declared
- *package*: (default) accessible from classes that are in the same package. If you don't declare a package name for your classes, they automatically belong to `default` package.
- *protected*: accessible from classes that are in the same package and from subclasses even if they belong to another package
- *public*: accessible by all

Unified Modeling Language (UML)

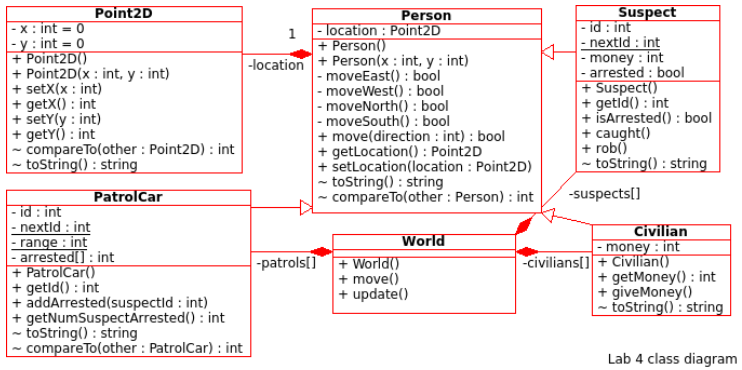
- UML is a graphical language for visualizing the architectural, behavioral and structural aspects of complex software systems.
- object-oriented systems are generally modeled using UML
- especially useful during the design phase to structure classes, interfaces, their relationships and division of responsibilities among them
- diagrams are the main tools in UML, divided in two groups:
  - structural: class diagram, object diagram etc.
  - behavioral: sequence diagram, collaboration diagram etc.
- a number of tools exist (Umbrella, Dia, Umlet etc.) for UML and some IDE's such as Eclipse have plugins that can build parts of the source code based on some UML diagrams.

Unified Modeling Language (Class Diagram)



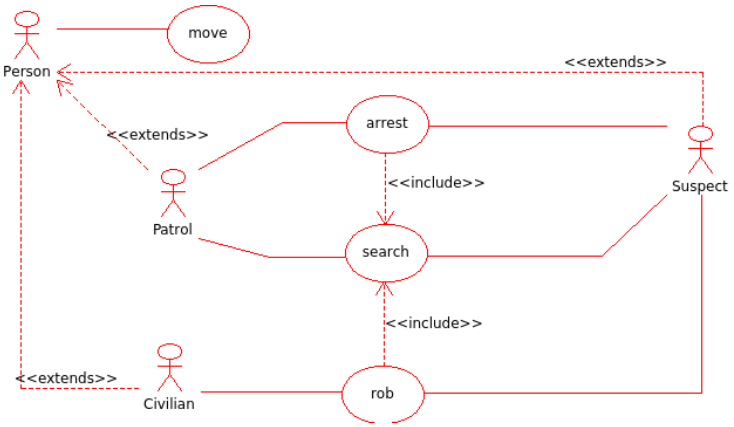
Lab 3 class diagram

Unified Modeling Language (Class Diagram 2)

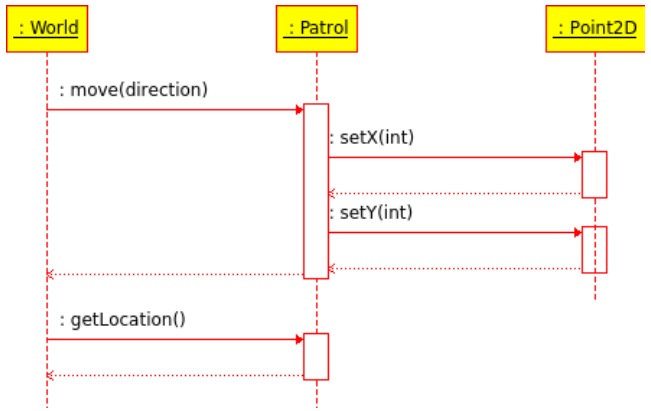


Lab 4 class diagram

Unified Modeling Language (Use Case Diagram)



Unified Modeling Language (Sequence Diagram)



review of OOP Concepts.

- Encapsulation: a language construct that facilitates the bundling of the data with the methods opertating on that data (e.g. decalaring data private and controlling the access to data with public methods)
- Inheritance: a way to reuse code of existing objects

```
public class A {
   public String getName() { return "A"; }
}
public class B extends A {
   pubic String getName() { return "B"; }
}
```

- Polymorphism: is the ability to create an object that has more than one form

```
A[] arrayA = new A[2];
arrayA[0] = new A();
arrayA[1] = new B();  // both an instance of A and B
```

- Dynamic binding: determining the 'true' type, therefore the behavior of an object at run-time.

```
for( A instanceA : arrayA )
   System.out.println(instanceA.getName());
```