**JAC444 - Lecture 3**

Object-Oriented Concepts

Segment 3 - Polymorphism

# Polymorphism

**class Point { int x; int y; void clear() { x = 0; y = 0 } }**

**class Pixel extends Point {** *Point* object

**Color color; public void clear() { super.clear();**

**color = null;** *Pixel* object

**} }**

**Pixel** extends both data and behavior **of its Point superclass.**

**All the Point code can be used by anywhere with a Pixel in hand.** A single object like **Pixel** could have many (poly) forms (-morph) It can be used as both a **Pixel** object and a **Point** object.

**Pixel**’s behavior extends **Point**’s behavior.

**Point point = new Pixel();**Implicit casting – Upcasting

a reference of extended class (**Pixel**) is assigned to a reference of the base class (**Point**)

# Constructor Order

* Each constructor has three phases:
  1. Invoke a superclass’s constructor.
  2. Initialize the fields using their initializers and any initialization block.

**0** for all numeric types, **false** for *boolean,* **\u0000**for *char,* **null** for references.

* 1. Execute the body of constructor.
* Each class has at least one constructor

If a class has no constructor the compiler adds the *default constructor*.

# Keyword: super

• Accessing fields and methods in superclass through object reference: **super**

**public class A { public void m() { System.out.println("In Superclass.");**

**} }**

**public class B extends A { // overrides m in the A class public void m() { super.m(); System.out.println("In Subclass"); }**

**public static void main(String[] args) { B x = new A();**

**x.m(); // what does it print? }**

**}**

**Constructors - super(); this();**

**class Rectangle extends Shape { int width = 0; int height = 0; Point origin;**

**Rectangle(Color c) {**

**super(c); //**super() superclass constructor invocation  **origin = new Point(); }**

**Rectangle (Color c, Point p) { this(c); //**this() explicit constructor invocation

**origin = p;**

**}**

**public move (Point origin) { this.origin = origin; //**this current object reference

**}**

**}**

# ObjectSuperClass

• At the top of the class hierarchy tree is the class **Object**

**protected Object clone() throws CloneNotSupportedException**  Creates and returns a copy of this object.

**public boolean equals(Object obj)**

Indicates whether some other object is "equal to" this one.

**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

**public final Class getClass()**

Returns the runtime class of an object.

**public int hashCode()**

Returns a hash code value for the object.

**public String toString()**

Returns a string representation of the object.

There are: **notify*,* notifyAll*, and* wait**methods for synchronizing activities in running Threads

# Final Classes / Methods

* A class can be declared as final with the declaration: **public final class X { …}**
* A class that is declared final cannot be subclassed Example: **java.lang.String**
* A method can be declared as final with the declaration:

**public class Y {**

**public final void m() {…}**

**}**

A method that is declared final cannot be overridden or hidden by subclasses

# Packages

* A *package* is a grouping of related types providing access protection and name space management

* Create a package with a **package** statement at the top of every source file
* Use **import** statement at the beginning of the file to work with package elements
* Conventions:
* Package names are written in all lowercase to avoid conflict with the names of classes or interfaces.
* The beginning of the package name must be a reversed Internet domain name

Example: **ca.senecacollege.ict**