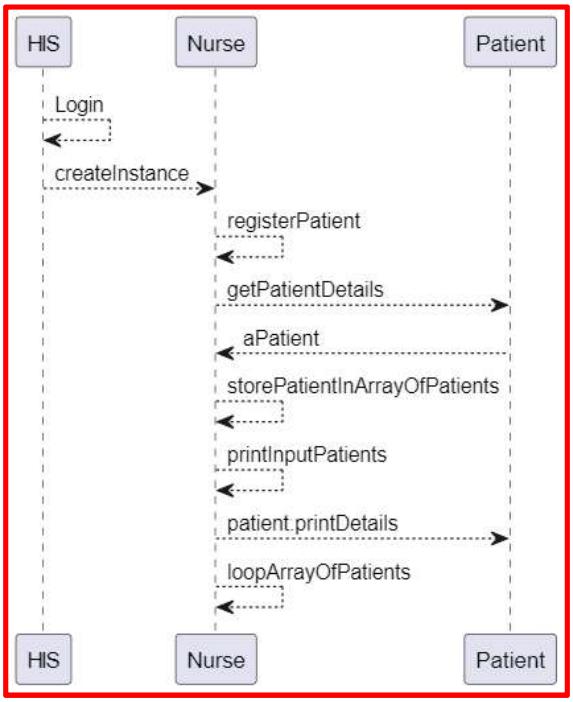
29th August 2022

IT, CT CN

Early risers

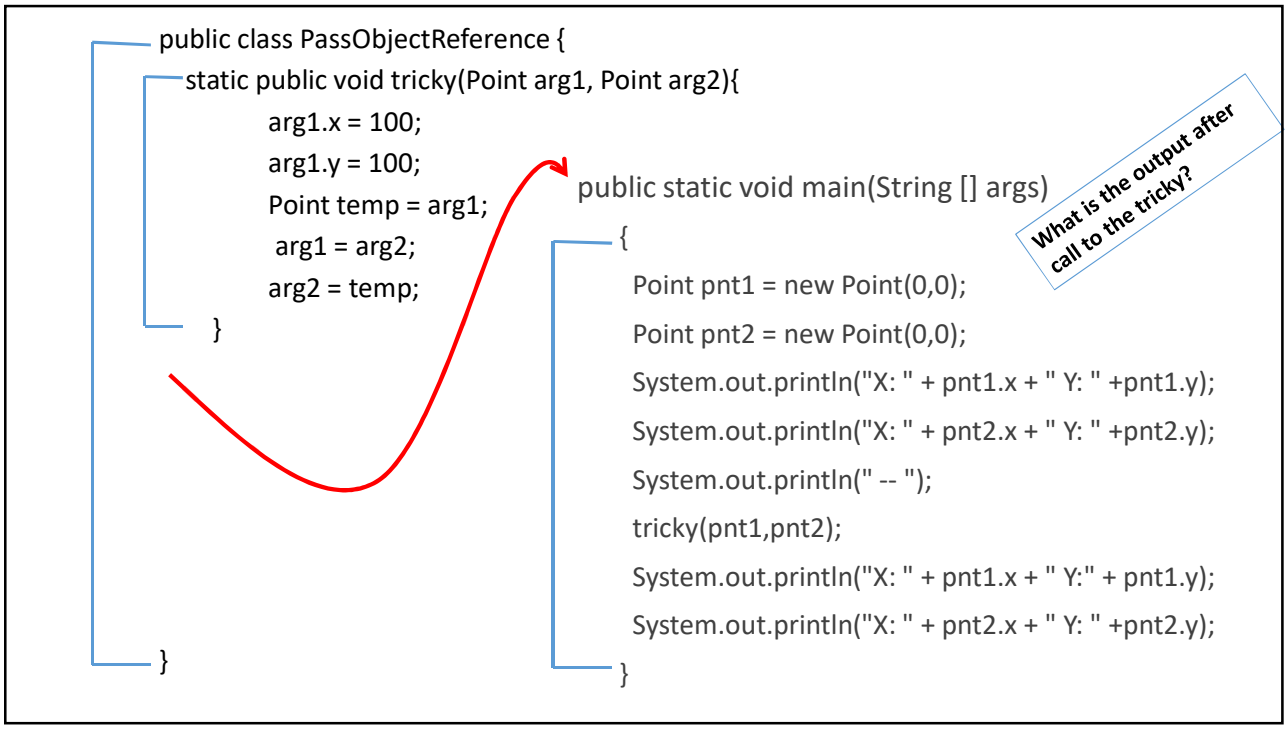
- StringWrapper class
  - Show difference of calling with an object reference and calling by value
- Recap of the exercise last week
  - HIS

Last week Monday!



Passing Object References

- In Java, we can pass a reference to an object (also called a "handle")as a parameter. We can then change something inside the object; we just can't change what object the handle refers to!



What passing object reference means

- The output:

  X: 0 Y: 0
  X: 0 Y: 0
  --
  X: 100 Y:100
  X: 0 Y: 0

- The method successfully alters the value of pnt1, even though it is passed by value; however, a swap of pnt1 and pnt2 fails!
- In the main() method, pnt1 and pnt2 are nothing more than object references. When you pass pnt1 and pnt2 to the tricky() method, Java passes the references by value just like any other parameter. This means the references passed to the method are actually *copies* of the original references.
- Java copies and passes the *reference* by value, not the object. Thus, method manipulation will alter the objects, since the references point to the original objects. But **since the references are copies, swaps will fai**l

THEORY

Modifiers, Inheritance and Polymorphism, Sample code

- **Modifiers**
  - Controlling Access to Members of a Class

Modifiers: Definition

- Modifiers are keywords that you add to objects definitions to change their meanings.
- Java language has a wide variety of modifiers, including the following –
  - Java Access Modifiers
  - Non Access Modifiers
- To use a modifier, you include its keyword in the definition of a class, method, or variable.
- E.g.
  - **public** class Person { … }
  - **public static** void main(String args[]) { … }

- Modifiers are used in the definition of:
  - a class,
  - method, or
  - variable
    static final double pi= 3.142

Access Modifiers

- Default Access Modifier - No Keyword
  - A variable or method declared without any access control modifier is available to any other class in the same package
- Private Access Modifier – private
  - Methods, variables, and constructors that are declared private can only be accessed within the declared class itself.
  - Using the private modifier is the main way that an object **encapsulates** itself and hides data from the outside world.
  - It is the most restrictive access level
  - Variables that are declared private can be accessed outside the class, if public getter methods are present in the class

```
package class7;


public class Person {
   private String name;

   public void setName(String aName){
      /* we possibly defined name as Private so that we can control
          values of name are all upper case */
      this.name=aName.toUpperCase();

   }

}
```

Imagine a getter method (getName(flag)) in this class that would allow flags to show the name as either:
1 (Uppercase) or 2 (lower case) or
3(Capitalize Each Word) or 4(Sentence case)

Access Modifiers

- *Default Access Modifier - No Keyword*
- *Private Access Modifier – private*
- Public Access Modifier - public
  - A class, fields, method, constructor, interface, etc. declared public can be accessed from any other class. Everything inside a public class can be accessed from any class belonging to the Java Universe.
  - However, if the public class we are trying to access is in a different package, then the public class still needs to be imported

public static void main(String[] arguments) { // ... }
- Is one of the most public methods in java

Access Modifiers

- *Default Access Modifier - No Keyword*
- *Private Access Modifier – private*
- *Public Access Modifier - public*
- Protected Access Modifier - protected
  - Variables, methods, and constructors, which are declared protected in a class can be accessed only by the classes in the same package or any subclass of this class (even if in another package)

- Summary
  - The following table shows the access to members permitted by each 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

REF: https://docs.oracle.com/javase/tutorial/java/javaOO/accesscontrol.html

Access Control and Inheritance

- The following rules for inherited methods are enforced –
  - Methods declared public in a superclass also must be public in all subclasses.
  - Methods declared protected in a superclass must either be protected or public in subclasses; they cannot be private.
  - Methods declared private are not inherited at all, so there is no rule for them.

Java - Non Access Modifiers

- Non-access modifiers are used to achieve many other functionalities:
  - The ***static*** modifier for creating class methods and variables.
  - The ***final*** modifier for finalizing the implementations of classes, methods, and variables.
  - The ***abstract*** modifier for creating abstract classes and methods.
  - The ***synchronized*** and ***volatile*** modifiers, which are used for threads.

- The Static Modifier
  - Static Variables
    - The *static* keyword is used to create variables that will exist independently of any instances created for the class. Only one copy of the static variable exists regardless of the number of instances of the class.
    - Static variables are also known as class variables. Local variables cannot be declared static.
  - Static Methods
    - The static keyword is used to create methods that will exist independently of any instances created for the class.
    - Static methods do not use any instance variables of any object of the class they are defined in. Static methods take all the data from parameters and compute something from those parameters, with no reference to variables.
    - Class variables and methods can be accessed using the class name followed by a dot and the name of the variable or method.

```
public class InstanceCounter {

private static int numInstances = 0;

  protected static int getCount() {
    return numInstances;
  }

  private static void addInstance() {
    numInstances++;
  }

  InstanceCounter() {
    InstanceCounter.addInstance();
  }

}
```

What is the output?

Output
Started with 0 instances
Created 500 instances

```
  public static void main(String[] arguments) {
    System.out.println("Starting with " +
InstanceCounter.getCount() + " instances");
    for (int i = 0; i < 500; ++i) {
      new InstanceCounter();
    }
    System.out.println("Created " +
InstanceCounter.getCount() + " instances");
  }
```

- The Final Modifier
  - Final Variables
    - A final variable can be explicitly initialized only once. A reference variable declared final can never be reassigned to refer to an different object.
    - However, the data within the object can be changed. So, the state of the object can be changed but not the reference.
    - With variables, the *final* modifier often is used with *static* to make the constant a class variable.
  - Final Methods
    - A final method cannot be overridden by any subclasses. This prevents a method from being modified in a subclass.
    - The main intention of making a method final would be that the content of the method should not be changed by any outsider.
  - Final Classes
    - The main purpose of using a class being declared as *final* is to prevent the class from being subclassed. If a class is marked as final then no class can inherit any feature from the final class.

- The abstract Modifier
  - Abstract Class
    - An abstract class can never be instantiated. If a class is declared as abstract then the sole purpose is for the class to be extended. A class cannot be both abstract and final (since a final class cannot be extended). If a class contains abstract methods then the class should be declared abstract. Otherwise, a compiler error will be thrown.
    - An abstract class may contain both abstract methods as well normal methods.
  - Abstract Methods
    - An abstract method is a method declared without any implementation. The methods body (implementation) is provided by the subclass. Abstract methods can never be final or strict.
    - Any class that extends an abstract class must implement all the abstract methods of the super class, unless the subclass is also an abstract class.
    - The abstract method ends with a semicolon.
      Example: ***public abstract void sample();***

- The Synchronized Modifier
  - The synchronized keyword used to indicate that a method can be accessed by only one thread at a time. The synchronized modifier can be applied with any of the four access level modifiers.
  - Viz:
    - *Default Access Modifier - No Keyword (in the package)*
    - *Private Access Modifier – private (in the class)*
    - *Public Access Modifier – public (in the java universe)*
    - *Protected Access Modifier – protected (in the package + subclasses – even if subclasses are in other packages)*

Relevant for this level of this course Please practice more and try the other modifiers as well

- Many Thanks!

Inheritance and Polymorphism

Class

- **Definition:**
  - A class is a descriptor of specified logical similarities between objects that are specified in the same way
  - **A class is a template from which individual objects are constructed when they are needed**
    - → An object is an instance of a class and is unique

Classes and Instance - 1

- All objects are *instances* of some *class*
- A Class is a description of a set of objects with similar:
  - features (attributes, operations, links);
  - semantics;
  - constraints (e.g. when and whether an object can be instantiated).

Classes and Instance - 2

- An object is an instance of some class
- So, instance = object
  - but also carries connotations of the class to which the object belongs
- Instances of a class are similar in their:
  - *Structure*: what they *know*, what information they hold, what links they have to other objects
  - *Behaviour*: what they *can do*

True or False?

- When two objects cannot be described by a simple set of features, they cannot belong to the same class

Think from a specific system point of view: Library system or a university management system

- All objects of a class share a common set of valid behaviors

Both are true!

Generalization and Specialization

Reflection Questions:

- What is the difference?
- How important is generalization in reality?

Generalization and Specialization

- Generalization occurs when there is a taxonomic relationship between two classes
  - Specification of one class is more general and applies also to the second class, while the specification of the second class is more specific and includes some details that do not apply to the first
  - An instance of the general class is also indirectly an instance of the more specific class

True or False ??

- An instance of the specific class is also indirectly an instance of the more general class
- An instance of the general class is also indirectly an instance of the more specific class

Answer

- An instance of the specific class is also indirectly an instance of the more general class
- ~~An instance of the general class is also indirectly an instance of the more specific class~~
- **Exercise:**
  - **Describe a case in a system to demonstrate the above statement**
- A general class represents the common characteristics shared by all specific classes
  - Specific classes should have fewer characteristics

- Practical uses of generalizations: Case of the TU-K GateSystem
  - Exercise to identify all the possible generalizations, feel free to extend the described system

- In the context of the ongoing modernization of TU-K, the University head of Security has approached you to design a simple system for the ***askaris*** manning the University gate. The idea is to automate / computerize what happens at the gate, starting with the Visitors-Book. The visitors book records all visitors to the University premises. At any given gate, a record of a visitor shows, among other things the following:
  - The officer who attended to the visitor, the date of visit, details of the visitor, destination point/office, the purpose/objective of the visit, mode of traveling used (and corresponding details), report from the visited person, gate used to exit, etc
  - After listening to the visitor, the security officer (***askari***) must indicate on the register ~~form~~ whether the purpose of visit is official, private, or returning resident. Official visit can be administrative office visit, lecturing, studying, working,
- etc

TU-K GateSystem

- Person
  - Staff
  - Student
    - Part-time lecturer
    - Full-time lecturer
  - Guest
- ModeOfTravel
  - Vehicle
  - Cycle
  - None

- Create properties and methods for person
  - Id, number, name, department, etc.
  - Abstract the common properties to Person and others to specific class
  - Methods
    - printDetails() method
    - identify(with University Reg. Number, with ID Number, with Personnel/Payroll Number, etc.)
    - ➔ Override methods
      - To provide capability for specialized functionalities (***Ref. previous session***)

Single and multiple inheritance

- In the inheritance hierarchy, classes become more specific and concrete with each new subclass. If you move from a subclass back up to a superclass, the classes become more general and less specific
- Class design should ensure that a superclass contains common features of its subclasses. Sometimes a superclass is so abstract that it cannot have any specific instances. Such a class is referred to as an ***abstract class***.

Single and multiple inheritance (Contd.)

- Sometimes it is necessary to derive a subclass from several classes. This capability is known as multiple inheritance.
- Java, however, does not allow multiple inheritance. A Java class may inherit directly from only one superclass. This restriction is known as single inheritance. If you use the extends keyword to define a subclass, it allows only one parent class. With interfaces, you can obtain the effect of multiple inheritance.

Abstract Classes

- GeometricObject was declared as the superclass for Circle and Rectangle in the preceding chapter. GeometricObject models common features of geometric objects.
- Both Circle and Rectangle contain the getArea() and getPerimeter() methods for computing the area and perimeter of a circle and a rectangle. Since you can compute areas and perimeters for all geometric objects, it is better to declare the getArea() and getPerimeter() methods in the GeometricObject class.
- However, these methods cannot be implemented in the GeometricObject class because their implementation is dependent on the specific type of geometric object. Such methods are referred to as abstract methods. After you declare the methods in GeometricObject, GeometricObject becomes an abstract class. The new GeometricObject class is shown below.

```
public abstract class GeometricObject {
    …
}
```

- /** Abstract method getArea */
  ```
  public abstract double getArea();
  ```
- /** Abstract method getPerimeter */
  ```
  public abstract double getPerimeter();
  ```

- Abstract classes are like regular classes with data and methods, but you cannot create instances of abstract classes using the new operator. An abstract method is a method signature without implementation. Its implementation is provided by the subclasses.
- A class that contains abstract methods must be declared abstract.

Interfaces

- An interface is a class-like construct that contains only constants and abstract methods. In many ways, an interface is similar to an abstract class, but an abstract class can contain variables and concrete methods as well as constants and abstract methods.
- To distinguish an interface from a class, Java uses the following syntax to declare an interface:

modifier interface InterfaceName

e.g. public interface AttendanceMarker{

…

}

- An interface is treated like a special class in Java. Each interface is compiled into a separate bytecode file, just like a regular class.
- As with an abstract class, you cannot create an instance from an interface using the new operator, but in most cases you can use an interface more or less the same way you use an abstract class
- For example, you can use an interface as a data type for a variable, as the result of casting, and so on

- A popular interface is the
  - ActionListener
    - The listener interface for receiving action events.
    - Defines an abstract method called ➔ **void actionPerformed(ActionEvent e)**
    - The class that is interested in processing an action event implements this interface, and the object created with that class is registered with a component, using the component's addActionListener method. When the action event occurs, that object's actionPerformed method is invoked.
    - REF: How we use it in a menu (JFrame class)

Polymorphism

Was introduced