Lecture 15

Polymorphism, Interface, Abstract classes

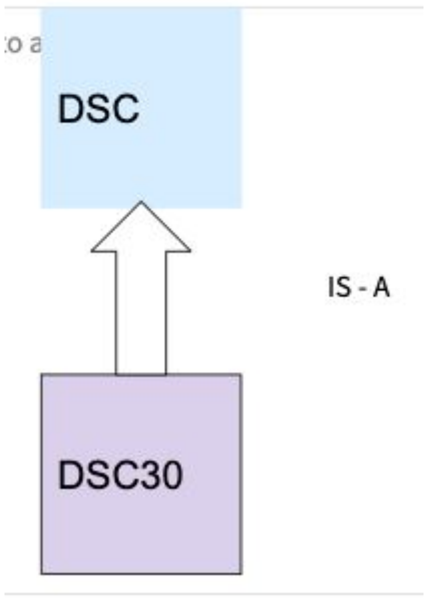
Power of "IS - A" (Polymorphism). Demo

Polymorphism is the ability of an object to take on ***many*** forms.

when a **parent** class reference is used to refer to a **child** class object.

```
DSC class2 = new DSC30();
```

DSC30 *IS-A* DSC



Variables in **Java** do not follow **polymorphism** and overriding is only applicable to methods but not to **variables**.

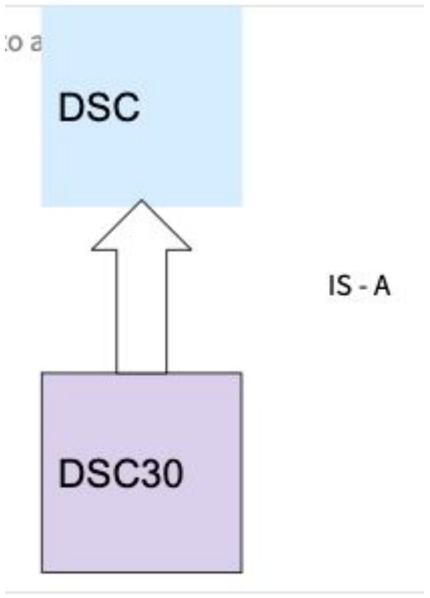
Polymorphic Array and Param Passing

```
Polymorphic Arrays
```

Power of “IS - A”. Demo

DSC30 class2 = new DSC();

Not OK.



Interface

Car class, revisited

Methods:

```
void accelerate( )

void takeDamage(int d)

void flip()

String blowUp( )

….
```

Interface

```
public interface Car {

    public void accelerate( );

    public void takeDamage(int d);

    public static void flip( );

    public String blowUp( );

}
```

- reference type: **Car c**
- Contains only *abstract* methods
  - No implementation, only method headers.
- interface may also contain constants, static methods, default methods (beyond the scope of this class)
- An interface does not contain any constructors.
- You cannot instantiate an interface.
  - **Car c = new Car () ←-wrong**
- It is *implemented* by a class.

Implements: Signing a contract

```
public interface Car {

    public void accelerate( );

    public void takeDamage(int d);

    public static void flip( );

    public String blowUp( );

}
```

```
public class Race implements Car {
                 int damage = 0;
                  // add one or more
constructors.
      public void accelerate( ){
           ...actual implementation...
      }
      public void takeDamage(int d){
            ...actual implementation...
      }
      public static void flip() {
            ...actual implementation...
      }
      public String blowUp( ){
            ...actual implementation...
      }
}
```

Implements: signing a contract

```
public interface Car {
     public void accelerate( );
     public void takeDamage(int d);
     public static void flip( );
     public String blowUp( );
}
```

What is a proper way to create an object?

A: Car c = new Car ()
B: Race r = new Race ()
C: Race r = new Car()
D: Car r = new Race ()
E: More than one possible answer

```
public class Race implements Car {
             int damage = 0;
              // add one or more
constructors.
     public void accelerate( ){
          ...actual implementation...
     }
     public void takeDamage(int d){
          ...actual implementation...
     }
     public static void flip() {
          ...actual implementation...
     }
     public String blowUp( ){
          ...actual implementation...
     }
}
```

Implements: Signing a Cont

```
public interface Car {
     public void accelerate( );
     public void takeDamage(int d);
     public static void flip( );
     public String blowUp( );
}
```

What is a proper way to create an object?

A: Car c = new Car ()
B: Race r = new Race ()
C: Race r = new Car()
D: Car r = new Race ()
E: More than one possible answer

```
public class Race implements Car {
             int damage = 0;
              // add one or more
constructors.
     public void accelerate( ){
         ...actual implementation...
     }
     public void takeDamage(int d){
          ...actual implementation...
     }
     public static void flip() {
          ...actual implementation...
     }
     public String blowUp( ){
          ...actual implementation...
     }
     public String test( ){
          ...actual implementation...
     }

}
```

One of the reasons why we use interface

ADT Implementers and Users
Implementers
Users
ADT Interface:
sets the rules of interaction
"We can implement the ADT however we want!"
"We can use the ADT however we want!"

https://www.java67.com/2014/02/what-is-actual-use-of-interface-in-java.html

Introducing: Abstract Classes

Abstract classes are an intermediate level between interfaces and classes.

- Cannot be instantiated.
- Can provide either **abstract** or **concrete** methods.
  - Use **abstract** keyword for abstract methods.
  - Use no keyword for concrete methods.
- Can provide variables (any kind).

Similarities

Differences

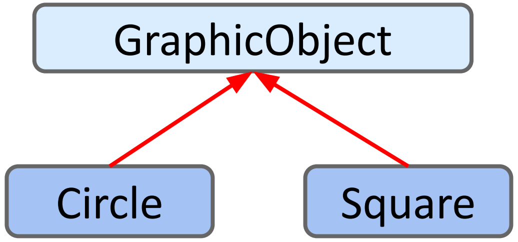
opposite of interfaces

```java
public abstract class GraphicObject {
    public int x, y;
    ...
    public void moveTo(int newX, int newY) { ... }
    public abstract void draw();
    public abstract void resize();
}
```

Example (From Oracle's Abstract Class Tutorial)

```java
public abstract class GraphicObject {
    public int x, y;
    ...
    public void moveTo(int newX, int newY) { ... }
    public abstract void draw();
    public abstract void resize();
}
```

```java
public class Circle extends GraphicObject {
    public void draw() { ... }
    public void resize() { ...  }
}
```



Implementations must override ALL abstract methods.

Question

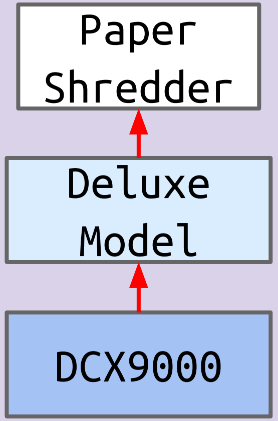
```
public interface PaperShredder {
   void shred(Document d);
   void shredAll(Document[] d);
}
```

```
public abstract class DeluxeModel
      implements PaperShredder {
   public int count = 0;
   public void count() { return count; }

   public shredAll(Document[] d) {
      for (int i = 0; i < d.length; d += 1) {
         shred(d);
      }
   }
   public abstract void connectToWifi();
}
```

How many abstract methods **must** DCX9000 override?

A. 0
B. 1
C. 2
D. 3



Summary: Abstract Classes vs. Interfaces

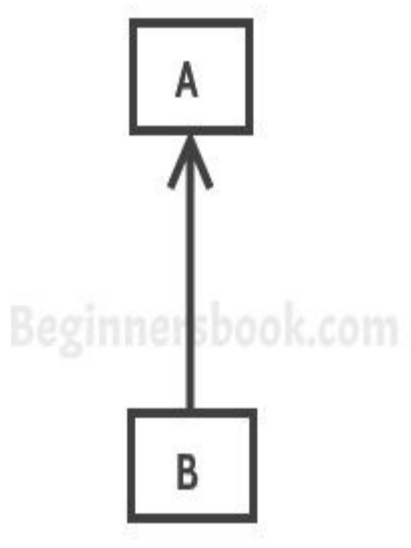
Interfaces:

- Primarily for interface inheritance. Limited implementation inheritance.
- **Classes can implement multiple interfaces.**

Abstract classes:

- Can do anything an interface can do, and more.
- **Subclasses only extend one abstract class.**

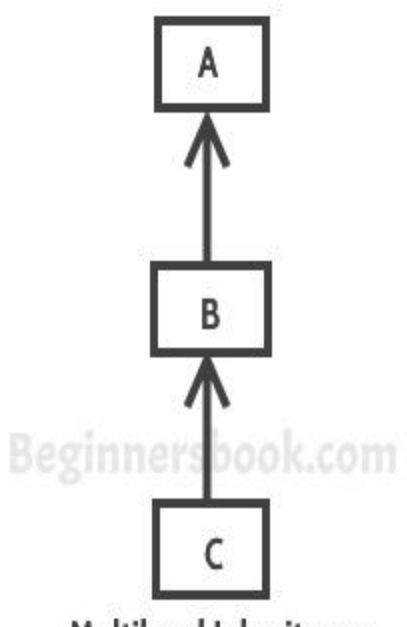
Types of inheritance: Single Inheritance



Single Inheritance

https://beginnersbook.com/2013/03/inheritance-in-java/

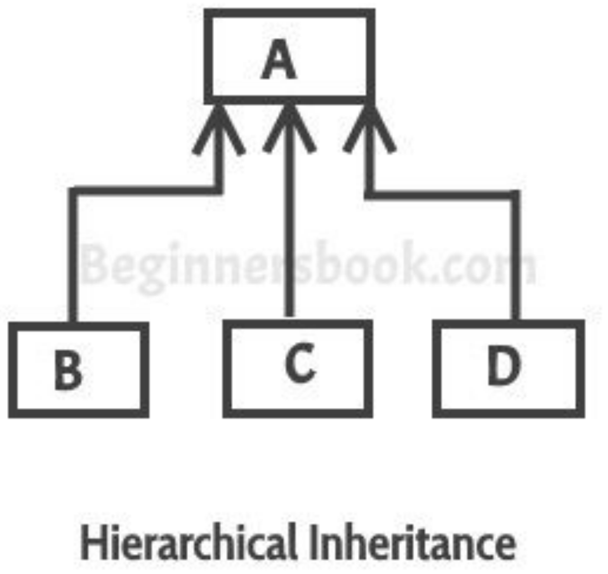
Types of inheritance: Multilevel inheritance



Multilevel Inheritance

https://beginnersbook.com/2013/03/inheritance-in-java/

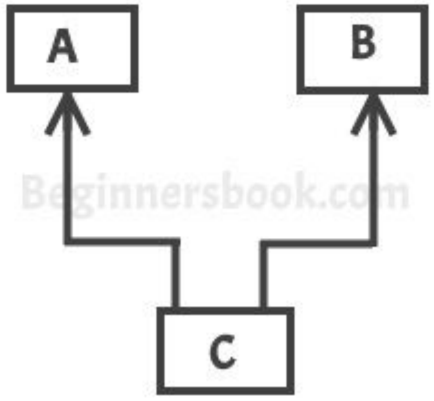
Types of inheritance: Hierarchical inheritance



https://beginnersbook.com/2013/03/inheritance-in-java/

Types of inheritance: Multiple Inheritance:

Not in Java! :(
Python has it



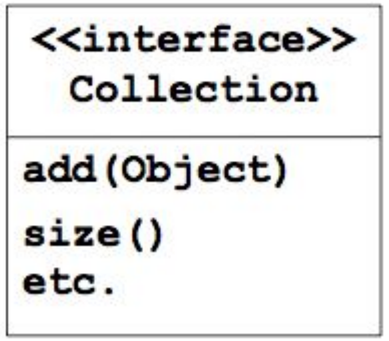
Multiple Inheritance

https://beginnersbook.com/2013/03/inheritance-in-java/

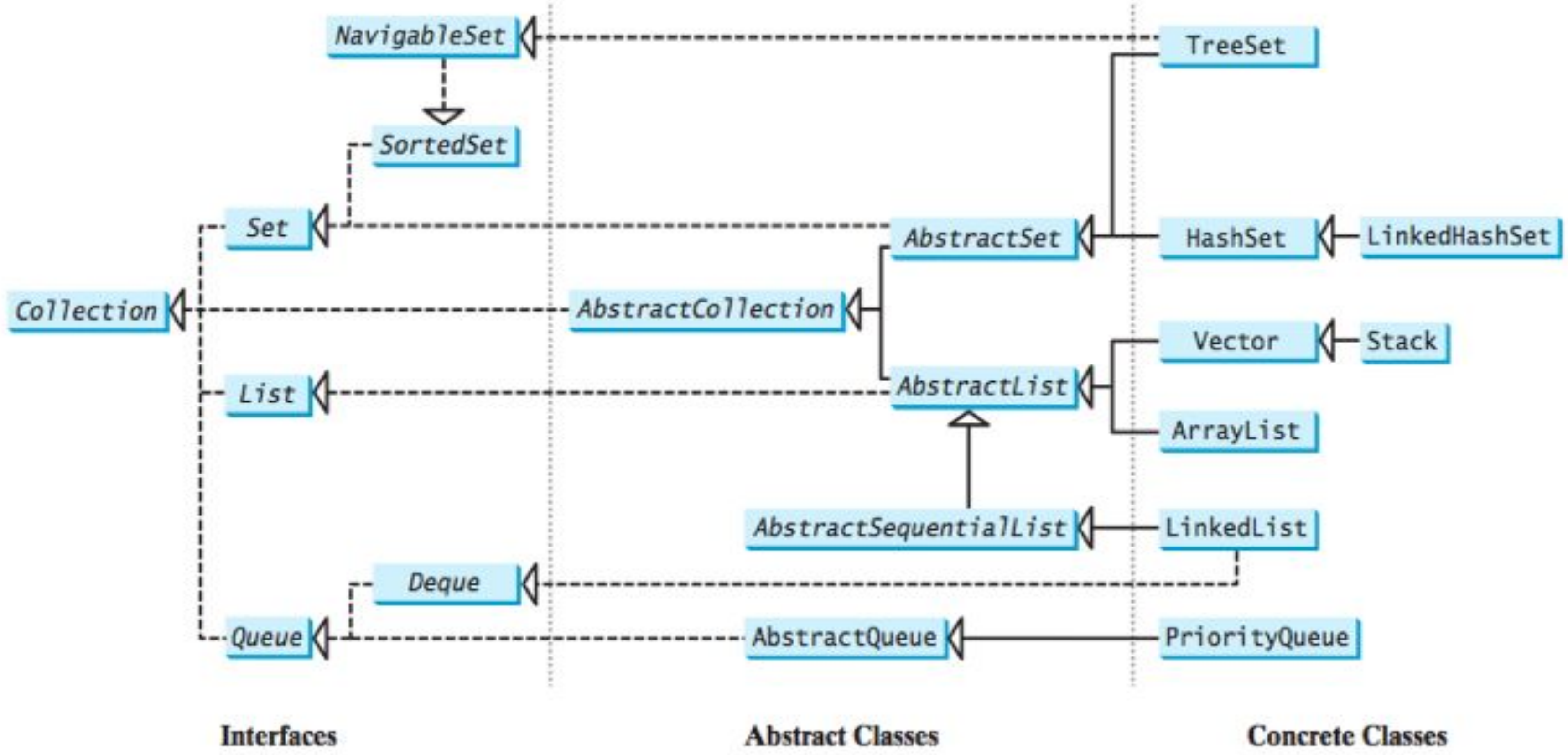
Collections

- Fundamentally, what we as programmers do with data is to store it and retrieve it and then operate on it.
- A **collection** is an ADT (Abstract Data Type) that contains data elements, and provides operations on them.
- There are different ways that elements can be collected:
  - Set, List, Sorted List…

All collections implement the interface `Collection`



A collection is a container that stores objects



Abstract List

- public class DoublyLinkedList<E> implements List<E> <--- **ideal**
- public class DoublyLinkedList<E> extends `AbstractList<E>`
- `AbstractList` provides *dummy* implementations for most methods in `List` interface.
- We can override its methods with our own!!

https://docs.oracle.com/javase/9/docs/api/java/util/AbstractList.html

Question 1

```
class Kids{
  public void like(){
    System.out.println("I like tag!");
  }
}
```

```
class Child1 extends Kids {
  @Override
  public void like(){
    System.out.println("I like dressup!");
  }
}
```

```
class Child2 extends Child1 {}
```

```
Child1 girl = new Child1();
girl.like();

Child2 boy = new Child2();
boy.like();
```

A: I like dressup!
I like tag!

B: I like dressup!
I like dressup!

C: I like tag!
I like tag!

D: I like tag!
I like dressup!

E: Error

Question 2

```
class Kids{
  public void like(){
    System.out.println("I like
tag!");
  }
}
```

```
class Child1 extends Kids {
  @Override
  private void like(){
    System.out.println("I like
dressup!");
  }
}
```

```
class Child2 extends Child1 {}
```

```
Child1 girl = new Child1();
girl.like();

Child2 boy = new Child2();
boy.like();
```

A: I like dressup!
I like tag!

B: I like dressup!
I like dressup!

C: I like tag!
I like tag!

D: I like tag!
I like dressup!

E: Error

Question 3

```
class Kids{
  protected void like(){
    System.out.println("I like
tag!");
  }
}
```

```
class Child1 extends Kids {
  @Override
  public void like(){
    System.out.println("I like
dressup!");
  }
}
```

```
class Child2 extends Child1 {}
```

```
Child1 girl = new Child1();
girl.like();

Child2 boy = new Child2();
boy.like();
```

A: I like dressup!
I like tag!

B: I like dressup!
I like dressup!

C: I like tag!
I like tag!

D: I like tag!
I like dressup!

E: Error

Question 4

```
class Kids{
  private void like(){
    System.out.println("I like
tag!");
  }
}
```

```
class Child1 extends Kids {
  @Override
  public void like(){
    System.out.println("I like
dressup!");

}
```

```
class Child2 extends Child1 {}
```

```
Child1 girl = new Child1();
girl.like();

Child2 boy = new Child2();
boy.like();
```

A: I like dressup!
I like tag!

B: I like dressup!
I like dressup!

C: I like tag!
I like tag!

D: I like tag!
I like dressup!

E: Error