Day 12

- In case of upcasting, using super class reference variable, we can access:
    1. Fields of super class inherited into sub class.
    2. Method of super class inherited into sub class.
    3. Overriden method of sub class.
- In case of upcasting, using super class reference variable, we can not access:

1. Fields of sub class
2. Non overriden method of sub class.

- In this case, we need to do downcasting.
- instanceof is an operator. In case of upcasting, it is used to check, super class reference is storing reference which sub class instance.

```
private static void acceptRecord(Product product) {
    System.out.print("Title     :   ");
    product.setTitle(sc.nextLine());
    System.out.print("price     :   ");
    product.setPrice(sc.nextFloat());
    if( product instanceof Book ) {
        Book book = (Book) product; //Downcasting
        System.out.print("Page Count    :   ");
        book.setPageCount(sc.nextInt());
    }else {
        Tape tape = (Tape) product; //Downcasting
        System.out.println("Play Time   :   ");
        tape.setPlayTime(sc.nextInt());
    }
}
```

- If downcasting fails, then JVM throws ClassCastException.

```
public static void main(String[] args) {
    Object obj = new Date( );
    Date date = (Date) obj;     //Downcasting : It will work
    String str = (String) obj;  //Downcasting : ClassCastException
}
```

Rules of method overriding

- Sub class overrides method of super class.

1. Access modifier in sub class method should be same or it should be wider than super class method.
2. Return type in sub class method should be same or it should be sub type. In other words it shout be covariant.

3. Method name, number of parameters and type of parameters in sub class method must be same.
4. Checked exception list in sub class method should be same or it should be sub set.

- Using above rules, if we redifine method in sub class then it is called method overriding.

We can not override following methods in sub class:

```
1. private methods
2. final method
3. static method
```

Override Annotation / Annotation Type

- Override is Annotation declared in java.lang package.
- It generates metadata(data which describes other data) for the compiler.
- If we do not follow rules of method overriding and if we use Override Annotation then Java compipler generates error.

Final method

- According to client's requirement, if implementation of super class method is logically 100% complete then we should declare super class method final.
- We can not override, final method in sub class.
- Final method inherit into sub class hence we can use it with instance of sub class.
- Example:
    - name() and ordinal() methods of java.lang.Enum class
    - Six methods of java.lang.Object class
        1. public final Class<?> getClass( );
        2. public final void wait( ) throws IE;
        3. public final void wait(long timeout) throws IE;
        4. public final void wait(long timeout, int nanos) throws IE;
        5. public final void notify( );
        6. public final void notifyAll( );
- We can declare overriden method final.

abstract method

- According to client's requirement, if implementation of super class method is logically 100% incomplete then we should declare super class method abstract.
- abstract is keyword in Java.
- We can not provide body to the abstract method.
    - A method of a class which is having a body is called concrete method(static/ non static).
    - A method of a class which is not having a body is called abstract method.
- If we declare method abstract then we must declare class abstract.
- It is mandatory to override abstract method in sub class otherwise sub class can be considered as abstract.

```
abstract class A{
    public abstract void f3( );
}
```

- Option 1

```
class B extends A{
    @Override
    public final void f3() {
        System.out.println("B.f3");
    }
}
```

- Option 2

```
abstract class B extends A{
}
```

Abstract class

- a class may/may not contain abstract method.
- Since implementation of abstract class is logically incomplete, we can not instantiate it. But we can create reference it.
- Example:
    1. java.lang.Number
    2. java.lang.Enum
    3. java.util.Calendar
    4. java.util.Dictionary

Final Class

- If implementation of a class is logically 100% complete then we should delcare such class final.
- We can instantiate final class but we can not extend it. In other words, we can not create sub class of final class.
- Example:
    1. java.lang.System
    2. java.lang.Math
    3. All wrapper classes
    4. java.lang.String, StringBuffer, StringBuilder
    5. java.util.Scanner
- we can not use abstract and final keyword together.

Object Class

- Object is non final and concrete class which is declared in java.lang package.
- java.lang.Object is ultimate base class / super cosmic base class / root of Java class hierarchy. In other words, all the classes(not interfaces) are directly or indirectly extended from java.lang.Object class.
- Object class do not extend any class or do not implement any interface.
- It doesn't contain nested type(Interface/class/enum/annotation).
- It doesn't contain any field.
- It contains only parameterless constructor(actually default).

```
Object o1 = new Object( );  //OK
Object o2 = new Object( "SunBeam" );    //NOT OK
```

- It constains 11 methods(5 non final + 6 final methods)
- Following are non final methods of java.lang.Object class.

1. public String toString( );
2. public boolean equals(Object obj);
3. public native int hashCode( );
4. protected native Object clone( )throws CloneNotSupportedException
5. protected void finalize( )throws Throwable

- Following are final methods of java.lang.Object class.

6. public final native Class<?> getClass( );
7. public final void wait( )throws InterruptedException
8. public final native void wait(long timeout)throws InterruptedException
9. public final void wait(long timeout, int nanos)throws InterruptedException
10. public final native void notity( );
11. public final native void notityAll( );