DAILY ASSESSMENT

Date:	**11-6-2020**	**Name:**	**Kavyashree m**
Course:	**VLSI**	**USN:**	**4al15ec036**
Topic:	**MOS transistor basics-II and III**	**Semester & Section:**	**8th A**
Github Repository:	**kavya**		

FORENOON SESSION DETAILS

Image of session

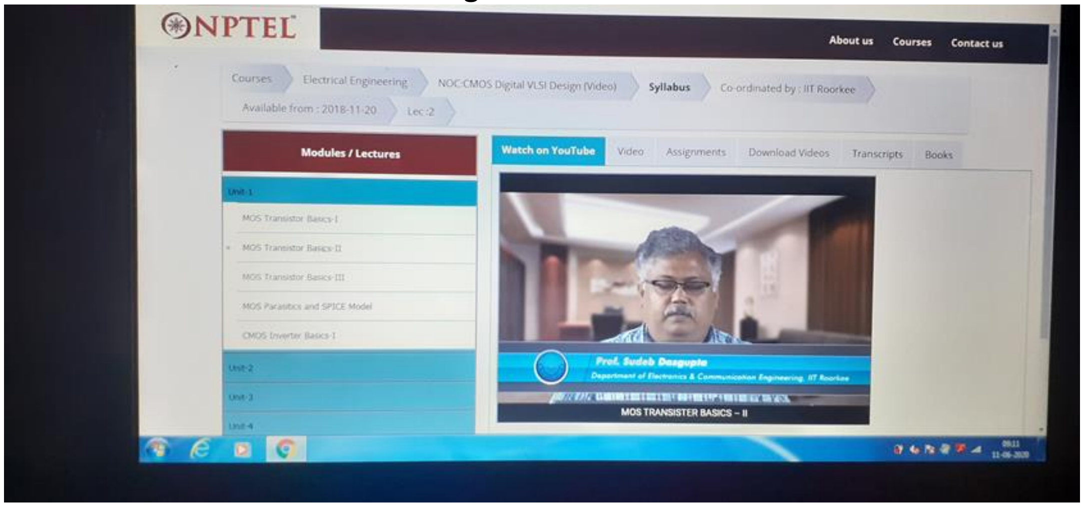


Fig : MOS transistor basics-II and III

MOS transistor basics-II and III

Current-voltage characteristics

n-channel Enhancement-type MOSFET

Figure 1a shows the transfer characteristics (drain-to-source current I_{DS} versus gate-to-source voltage V_{GS}) of n-channel Enhancement-type MOSFETs. From this, it is evident that the current through the device will be zero until the V_{GS} exceeds the value of threshold voltage V_T. This is because under this state, the device will be void of channel which will be connecting the drain and the source terminals. Under this condition, even an increase in V_{DS} will result in no current flow as indicated by the corresponding output characteristics (I_{DS} versus V_{DS}) shown by Figure 1b. As a result this state represents nothing but the cut-off region of MOSFET's operation.

Next, once V_{GS} crosses V_T, the current through the device increases with an increase in I_{DS} initially (Ohmic region) and then saturates to a value as determined by the V_{GS} (saturation region of operation) i.e. as V_{GS} increases, even the saturation current flowing through the device also increases. This is evident by Figure 1b where I_{DSS2} is greater than I_{DSS1} as $V_{GS2} > V_{GS1}$, I_{DSS3} is greater than I_{DSS2} as $V_{GS3} > V_{GS2}$, so on and so forth. Further, Figure 1b also shows the locus of pinch-off voltage (black discontinuous curve), from which V_P is seen to increase with an increase in V_{GS}.

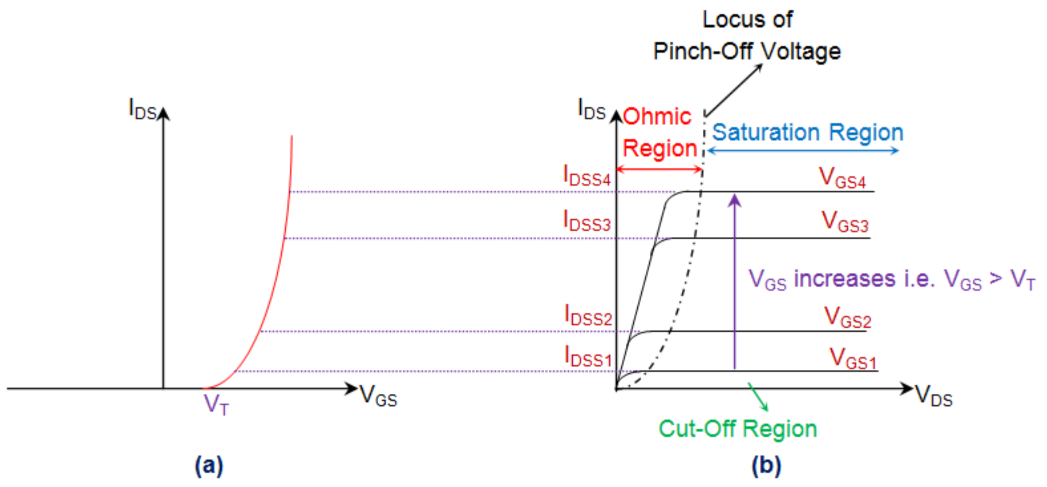


Figure 1 *n*-**Channel Enhancement type MOSFET (a) Transfer Characteristics (b) Output Characteristics**

p-channel Enhancement-type MOSFET

Figure 2a shows the transfer characteristics of p-type enhancement MOSFETs from which it is evident that I_{DS} remains zero (cutoff state) untill V_{GS} becomes equal to $-V_T$. This is because, only then the channel will be formed to connect the drain terminal of the device with its source terminal. After this, the I_{DS} is seen to increase in reverse direction (meaning an increase in I_{SD}, signifying an increase in the device current which will flow from source to drain) with the decrease in the value of V_{DS}. This means that the device is functioning in its ohmic region wherein the current through the device increases with an increase in the applied voltage (which will be V_{SD}).

However as V_{DS} becomes equal to $-V_P$, the device enters into saturation during which a saturated amount of current (I_{DSS}) flows through the device, as decided by the value of V_{GS}. Further it is to be noted that the value of saturation current flowing through the device is seen to increase as the V_{GS} becomes more and more negative i.e. saturation current for V_{GS3} is greater than that for V_{GS2} and that in the case of V_{GS4} is much greater than both of them as V_{GS3} is more negative than V_{GS2} while V_{GS4} is much more negative when compared to either of them (Figure 2b). In addition, from the locus of the pinch-off voltage it is also clear that as V_{GS} becomes more and more negative, even the negativity of V_P also increases.

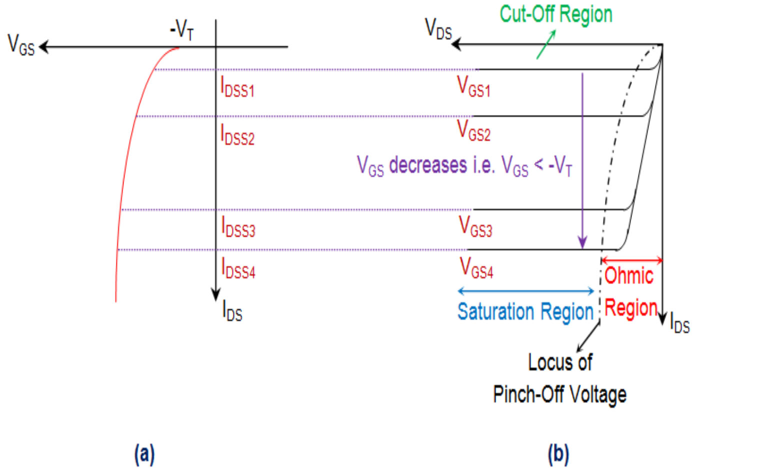


Figure 2 *p*-Channel Enhancement type MOSFET (a) Transfer Characteristics (b) Output Characteristics

n-channel Depletion-type MOSFET

The transfer characteristics of n-channel depletion MOSFET shown by Figure 3a indicate that the device has a current flowing through it even when V_{GS} is 0V. This indicates that these devices conduct even when the gate terminal is left unbiased, which is further emphasized by the V_{GS0} curve of Figure 3b. Under this condition, the current through the MOSFET is seen to increase with an increase in the value of V_{DS} (Ohmic region) untill V_{DS} becomes equal to pinch-off voltage V_P. After this, I_{DS} will get saturated to a particular level I_{DSS} (saturation region of operation) which increases with an increase in V_{GS} i.e. $I_{DSS3} > I_{DSS2} > I_{DSS1}$, as $V_{GS3} > V_{GS2} > V_{GS1}$. Further, the locus of the pinch-off voltage also shows that V_P increases with an increase in V_{GS}. However it is to be noted that, if one needs to operate these devices in cut-off state, then it is required to make V_{GS} negative and once it becomes equal to $-V_T$, the conduction through the device stops ($I_{DS} = 0$) as it gets deprived of its n-type channel (Figure 3a).

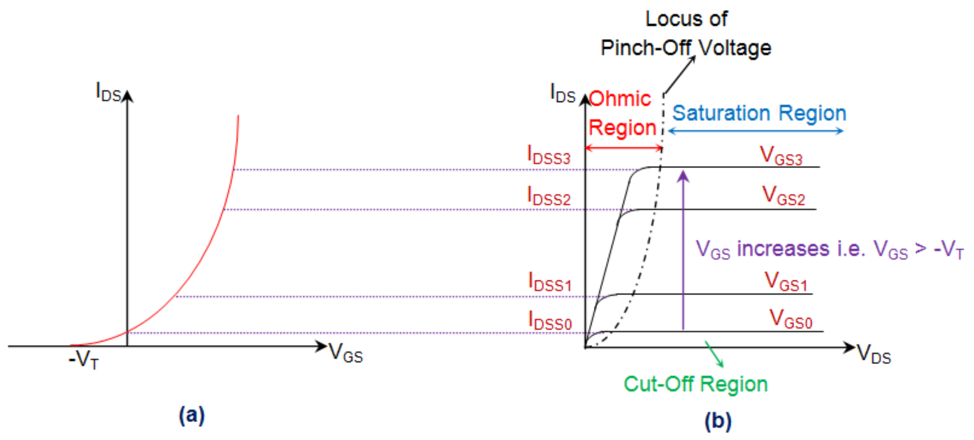


Figure 3 *n*-**Channel Depletion type MOSFET (a) Transfer Characteristics (b) Output Characteristics**

p-channel Depletion-type MOSFET

The transfer characteristics of p-channel depletion mode MOSFETs (Figure 4a) show that these devices will be normally ON, and thus conduct even in the absence of V_{GS}. This is because they are characterized by the presence of a channel in their default state due to which they have non-zero I_{DS} for $V_{GS} = 0V$, as indicated by the V_{GS0} curve of Figure 4b. Although the value of such a current increases with an increase in V_{DS} initially (ohmic region of operation), it is seen to saturate once the V_{DS} exceeds V_P (saturation region of operation). The value of this saturation current is determined by the V_{GS}, and is seen to increase in negative direction as V_{GS} becomes more and more negative. For example, the saturation current for V_{GS3} is greater than that for V_{GS2} which is however greater when compared to that for V_{GS1}. This is because V_{GS2} is more negative when compared to V_{GS1}, and V_{GS3} is much more negative when compared to either of them. Next, one can also note from the locus of pinch-off point that even V_P starts to become more and more negative as the negativity associated with the V_{GS} increases. Lastly, it is evident from Figure 4a that inorder to switch these devices OFF, one needs to increase V_{GS} such that it becomes equal to or greater than that of the threshold voltage V_T. This is because, when done so, these devices will be deprived of their p-type channel, which further drives the MOSFETs into their cut-off region of operation.

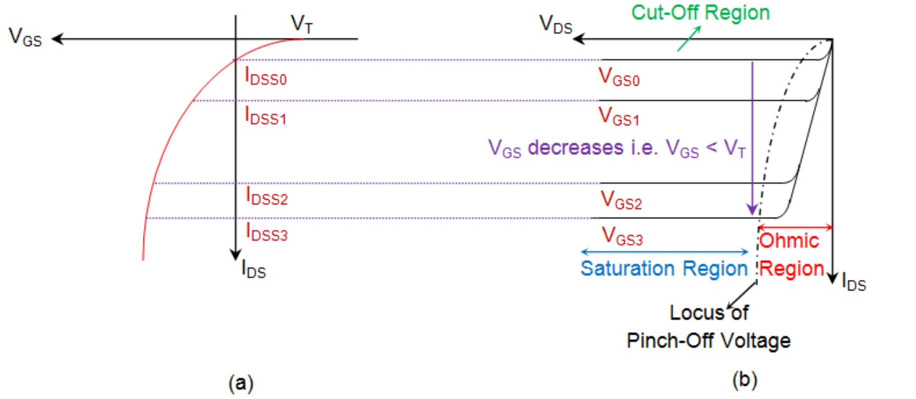


Figure 4 *p*-Channel Depletion type MOSFET (a) Transfer Characteristics (b) Output Characteristics

Short-channel effect

short-channel effects occur in MOSFETs in which the channel length is comparable to the depletion layer widths of the source and drain junctions. These effects include, in particular, drain-induced barrier lowering, velocity saturation, Quantum confinement and hot carrier degradation

Second order effects

channel length modulation

it is a shortening of the length of the inverted channel region with increase in drain bias for large drain biases. The result of CLM is an increase in current with drain bias and a reduction of output resistance. Channel length modulation occurs in all field effect transistors, not just MOSFETs.

To understand the effect, first the notion of pinch-off of the channel is introduced. The channel is formed by attraction of carriers to the gate, and the current drawn through the channel is nearly a constant independent of drain voltage in saturation mode. However, near the drain, the gate and drain jointly determine the electric field pattern. Instead of flowing in a channel, beyond the pinch-off point the carriers flow in a subsurface pattern made possible because the drain and the gate both control the current. In the figure at the right, the channel is indicated by a dashed line and becomes weaker as the drain is approached, leaving a gap of uninverted silicon between the end of the formed inversion layer and the drain (the pinch-off region).

Body effect

Body effect refers to the change in the transistor threshold voltage (VT) resulting from a voltage difference between the transistor source and body.Because the voltage difference between the source and body affects the VT, the body can be thought of as a second gate that helps determine how the transistor turns on and off.

Body bias involves connecting the transistor bodies to a bias network in the circuit layout rather than to power or ground.The body bias can be supplied from an external (off-chip) source or an internal (on-chip) source.

Types of device scaling

Velocity saturation

As devices are reduced in size, the electric field typically also increases and the carriers in the channel have an increased velocity. However at high fields there is no longer a linear relation between the electric field and the velocity as the velocity gradually saturates reaching the saturation velocity. This velocity saturation is caused by the increased scattering rate of highly energetic electrons, primarily due to optical phonon emission. This effect increases the transit time of carriers through the channel. In sub-micron MOSFETs one finds that the average electron velocity is larger than in bulk material so that velocity saturation is not quite as much of a restriction as initially thought.

Punch through

Punch through in a MOSFET is an extreme case of channel length modulation where the depletion layers around the drain and source regions merge into a single depletion region. The field underneath the gate then becomes strongly dependent on the drain-source voltage, as is the drain current. Punch through causes a rapidly increasing current with increasing drain-source voltage. This effect is undesirable as it increases the output conductance and limits the maximum operating voltage of the device.

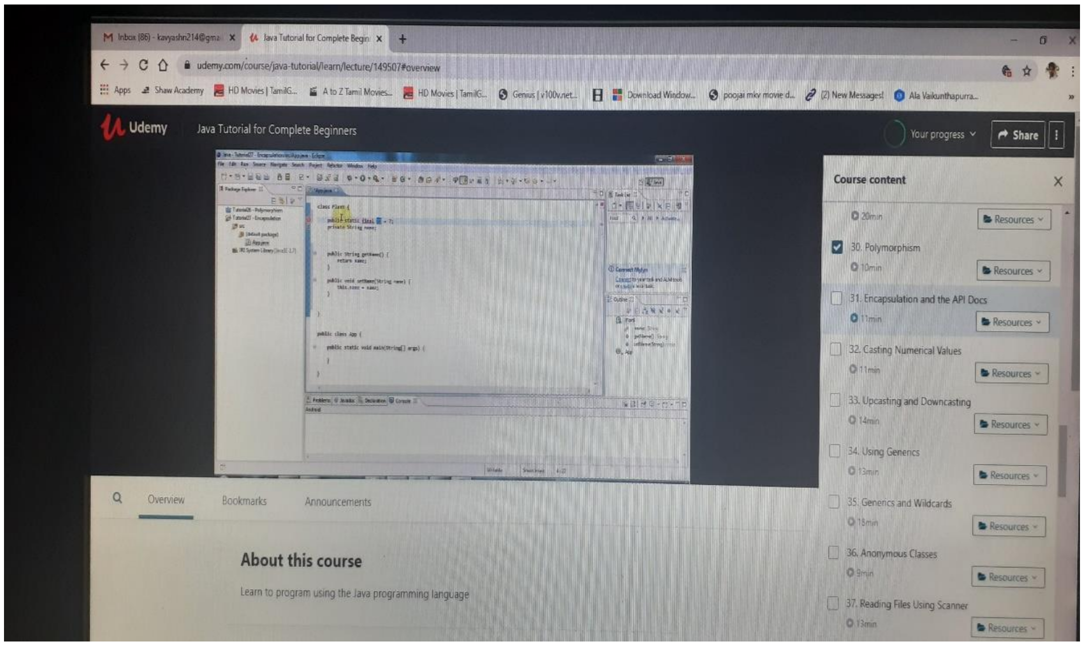
Drain induced barrier lowering

Drain induced barrier lowering (DIBL) is the effect the drain voltage on the output conductance and measured threshold voltage. This effect occurs in devices where only the gate length is reduced without properly scaling the other dimensions. It is observed as a variation of the measured threshold voltage with reduced gate length. The threshold variation is caused by the increased current with increased drain voltage as the applied drain voltage controls the inversion layer charge at the drain, thereby competing with the gate voltage. This effect is due to the two-dimensional field distribution at the drain end and can typically be eliminated by properly scaling the drain and source depths while increasing the substrate doping density.

AFTERNOON SESSION DETAILS

Date:	**11-6-2020**	**Name:**	**Kavyashree m**
Course:	**Java**	**USN:**	**4al15ec036**
Topic:	**The tostring method,inheritance,packages,interfaces,public,private and protected,polymorphism,encapsulation,casting numerical values,upcasting and downcasting,using generics**	**Semester & Section:**	**8th A**
Github Repository:	**kavya**		

Image of session


Udemy
Java Tutorial for Complete Beginners
Your progress
Share
Course content
20min
30. Polymorphism
10min
31. Encapsulation and the API Docs
11min
32. Casting Numerical Values
11min
33. Upcasting and Downcasting
14min
34. Using Generics
13min
35. Generics and Wildcards
18min
36. Anonymous Classes
9min
37. Reading Files Using Scanner
13min
Resources
Overview
Bookmarks
Announcements
About this course
Learn to program using the Java programming language

Java

Java toString() method

If you want to represent any object as a string, toString() method comes into existence.The toString() method returns the string representation of the object.

If you print any object, java compiler internally invokes the toString() method on the object. So overriding the toString() method, returns the desired output, it can be the state of an object etc. depends on your implementation.

Advantage of Java toString() method

By overriding the toString() method of the Object class, we can return values of the object, so we don't need to write much code.

Understanding problem without toString() method

Let's see the simple code that prints reference.

```
class Student{
 int rollno;
 String name;
 String city;

 Student(int rollno, String name, String city){
 this.rollno=rollno;
 this.name=name;
 this.city=city;
 }

 public static void main(String args[]){
   Student s1=new Student(101,"Raj","lucknow");
   Student s2=new Student(102,"Vijay","ghaziabad");

   System.out.println(s1);//compiler writes here s1.toString()
   System.out.println(s2);//compiler writes here s2.toString()
 }
}
```

Output:
```
Student@1fee6fc
Student@1eed786
```

Inheritance

Inheritance in Java is a mechanism in which one object acquires all the properties and behaviors of a parent object. It is an important part of OOPs (Object Oriented programming system).

The idea behind inheritance in Java is that you can create new classes that are built upon existing classes. When you inherit from an existing class, you can reuse methods and fields of the parent class. Moreover, you can add new methods and fields in your current class also.

Inheritance represents the IS-A relationship which is also known as a parent-child relationship.

Packages

A java package is a group of similar types of classes, interfaces and sub-packages.

Package in java can be categorized in two form, built-in package and user-defined package.There are many built-in packages such as java, lang, awt, javax, swing, net, io, util, sql etc.

Interfaces

An interface is a reference type in Java. It is similar to class. It is a collection of abstract methods. A class implements an interface, thereby inheriting the abstract methods of the interface.

Along with abstract methods, an interface may also contain constants, default methods, static methods, and nested types. Method bodies exist only for default methods and static methods.

Writing an interface is similar to writing a class. But a class describes the attributes and behaviors of an object. And an interface contains behaviors that a class implements.

Public,Private and Protected

Public access modifier

The members, methods and classes that are declared public can be accessed from anywhere. This modifier doesn’t put any restriction on the access.

public access modifier example in java

Lets take the same example that we have seen above but this time the method addTwoNumbers() has public modifier and class Test is able to access this method without even extending the Addition class. This is because public modifier has visibility everywhere.

Addition.java

```
package abcpackage;

public class Addition {

   public int addTwoNumbers(int a, int b){
         return a+b;
   }
}
```

Test.java

```
package xyzpackage;
import abcpackage.*;
class Test{
   public static void main(String args[]){
      Addition obj = new Addition();
```

```
    System.out.println(obj.addTwoNumbers(100, 1));
  }
}
```

Output:

```
101
```

Private access modifier

The scope of private modifier is limited to the class only.

1. Private Data members and methods are only accessible within the class
2. Class and Interface cannot be declared as private
3. If a class has private constructor then you cannot create the object of that class from outside of the class.

Let’s see an example to understand this:

Private access modifier example in java

This example throws compilation error because we are trying to access the private data member and method of class ABC in the class Example. The private data member and method are only accessible within the class.

```
class ABC{
  private double num = 100;
  private int square(int a){
        return a*a;
  }
}
public class Example{
  public static void main(String args[]){
        ABC obj = new ABC();
        System.out.println(obj.num);
        System.out.println(obj.square(10));
  }
}
```

Output:

```
Compile - time error
```

Protected Access Modifier

Protected data member and method are only accessible by the classes of the same package and the subclasses present in any package. You can also say that the protected access modifier is similar to default access modifier with one exception that it has visibility in sub classes. Classes cannot be declared protected. This access modifier is generally used in a parent child relationship.

Protected access modifier example in Java

In this example the class Test which is present in another package is able to call the addTwoNumbers() method, which is declared protected. This is because the Test class extends class Addition and the protected modifier allows the access of protected members in subclasses (in any packages).

Addition.java

```
package abcpackage;
public class Addition {

  protected int addTwoNumbers(int a, int b){
        return a+b;
  }
}
```

Test.java

```
package xyzpackage;
```

```
import abcpackage.*;
class Test extends Addition{
   public static void main(String args[]){
          Test obj = new Test();
          System.out.println(obj.addTwoNumbers(11, 22));
   }
}
```

Output:

```
33
```

Polymorphism

Polymorphism is the ability of an object to take on many forms. The most common use of polymorphism in OOP occurs when a parent class reference is used to refer to a child class object.

Any Java object that can pass more than one IS-A test is considered to be polymorphic. In Java, all Java objects are polymorphic since any object will pass the IS-A test for their own type and for the class Object.

It is important to know that the only possible way to access an object is through a reference variable. A reference variable can be of only one type. Once declared, the type of a reference variable cannot be changed.

The reference variable can be reassigned to other objects provided that it is not declared final. The type of the reference variable would determine the methods that it can invoke on the object.

A reference variable can refer to any object of its declared type or any subtype of its declared type. A reference variable can be declared as a class or interface type.

Encapsulation

Encapsulation is one of the four fundamental OOP concepts. The other three are inheritance, polymorphism, and abstraction.

Encapsulation in Java is a mechanism of wrapping the data (variables) and code acting on the data (methods) together as a single unit. In encapsulation, the variables of a class will be hidden from other classes, and can be accessed only through the methods of their current class. Therefore, it is also known as data hiding.

To achieve encapsulation in Java −

- Declare the variables of a class as private.
- Provide public setter and getter methods to modify and view the variables values.

Casting numerical values

Type casting is when you assign a value of one primitive data type to another type.

In Java, there are two types of casting:

- Widening Casting (automatically) - converting a smaller type to a larger type size byte -> short -> char -> int -> long -> float -> double

- Narrowing Casting (manually) - converting a larger type to a smaller size type double -> float -> long -> int -> char -> short -> byte

Upcasting and downcasting

Upcasting (Generalization or Widening) is casting to a parent type in simple words casting individual type to one common type is called upcasting while downcasting (specialization or narrowing) is casting to a child type or casting common type to individual type.

Using generics

Java Generic methods and generic classes enable programmers to specify, with a single method declaration, a set of related methods, or with a single class declaration, a set of related types, respectively.

Generics also provide compile-time type safety that allows programmers to catch invalid types at compile time.

Using Java Generic concept, we might write a generic method for sorting an array of objects, then invoke the generic method with Integer arrays, Double arrays, String arrays and so on, to sort the array elements.

Generic Methods

You can write a single generic method declaration that can be called with arguments of different types. Based on the types of the arguments passed to the generic method, the compiler handles each method call appropriately. Following are the rules to define Generic Methods −

- All generic method declarations have a type parameter section delimited by angle brackets (< and >) that precedes the method's return type (< E > in the next example).
- Each type parameter section contains one or more type parameters separated by commas. A type parameter, also known as a type variable, is an identifier that specifies a generic type name.
- The type parameters can be used to declare the return type and act as placeholders for the types of the arguments passed to the generic method, which are known as actual type arguments.
- A generic method's body is declared like that of any other method. Note that type parameters can represent only reference types, not primitive types (like int, double and char).

Example

Following example illustrates how we can print an array of different type using a single Generic method −

```
public class GenericMethodTest {
   // generic method printArray
   public static < E > void printArray( E[] inputArray ) {
      // Display array elements
      for(E element : inputArray) {
         System.out.printf("%s ", element);
      }
      System.out.println();
   }

   public static void main(String args[]) {
      // Create arrays of Integer, Double and Character
      Integer[] intArray = { 1, 2, 3, 4, 5 };
      Double[] doubleArray = { 1.1, 2.2, 3.3, 4.4 };
      Character[] charArray = { 'H', 'E', 'L', 'L', 'O' };

      System.out.println("Array integerArray contains:");
      printArray(intArray);   // pass an Integer array

      System.out.println("\nArray doubleArray contains:");
      printArray(doubleArray);   // pass a Double array

      System.out.println("\nArray characterArray contains:");
      printArray(charArray);   // pass a Character array
   }
}
```

This will produce the following result −

Output

```
Array integerArray contains:
1 2 3 4 5

Array doubleArray contains:
1.1 2.2 3.3 4.4

Array characterArray contains:
H E L L O
```