letter size, orientation, and margin ... conversion to take place. And that's all there is. Save the converted PDF ...
★★★★★ Rating 4.5 · 640,691 votes
PNG to PDF · How To Convert JPG To PDF · Convert JPG to Word Online

https://www.ilovepdf.com › jpg_to_pdf
Convert JPG to PDF. Images JPG to PDF online - iLovePDF
Convert JPG images to PDF, rotate them or set a page margin. Convert JPG to PDF online, easily and free.

https://jpg2pdf.com
JPG to PDF – Convert JPG Images to PDF Documents Online
This free online JPG to PDF converter allows to combine multiple images into a single PDF document. Besides JPG/JPEG, this tool supports conversion of PNG ...

PAGE NO.
DATE / /20

Interpreter	Compiler.
(1) Translates program one statement at a time	(1) Scans the entire program and translates is as machine code
(2) Interpreter usally take Less amount of time to analyze the Source code	(2) Compiler usally take a large amount of time to analyze the source code.
(3) the overall execution time is comparatively slower than compilers.	(3) the overall execution time is comparatively faster than interpreters.
(4) No object code is generated, hence are memory efficient.	(4) Generates object code which further requires linking, hence requires more memory.
(5) It requires source code for later execution.	(5) It does not require source code for later execution.
(6) keeps translating the program continously till the first error is confronted if any error is spotted, it stops working and hence debugging become easy	(6) A compiler generates the error message only After it scans the complete program and hence debugging is relatively harder while working with a compiler.
(7) compiler is based on translation linking-loading model, whereas Interpreter is based on Interpreter method.	(7). Interpreter model. is based on Interpreter method compiler is based on translating linker -loading model
(8) Interpreted programs can run on computers that have the corresponding Interpreter.	(8) you cannot chang the program without going back to the source code.
(9) It is best suited for the program and development environment	(9) It is best suited the production environment.
10) mostly used in ~~c, c++~~, ~~c#~~, Scala, ~~Java~~ PHP, Perl, Ruby.	(10) mostly used in c, c++, c#, Scala, Java.

When
After
Befo
objec
a c
Co
Co
in

1. tight coupling :

When an object creates the object to be used, then it is a tight coupling situation. As the main object creates the object itself, this object can not be changed from outside world easily marked it as tightly coupled objects.

examples :

```
            public class lg {
  public static void main(String args[]) {
    tv a = new tv();
    a.display();
  }
}

class tv {
  dish b;
  public tv() {

    b = new dish();
  }

  public void display() {
    System.out.println("tv is on");
    b.display();
  }
}

class B {
  public B(){}
  public void display() {
    System.out.println("dish is active channels are comming");
  }
}
```

EXAMPLE :

```
  class Subject {
  Topic t = new Topic();
  public void startReading()
  {
     t.understand();
  }
}
class Topic {
  public void understand()
  {
     System.out.println("no doubts");
  }
}
```

EXAMPLE :

```
  class height
{
   public static void main(String args[])
   {
      height b = new hight(5.8f);
      System.out.println(b.volume);
   }
}
class age
{
    public float height;
    age(float h)
    {
       this.height = h;
    }
}
```

LOOSE COUPLE ;

When an object gets the object to be used from the outside, then it is a loose coupling situation. As the main object is merely using the object, this object can be changed from the outside world easily marked it as loosely coupled objects.

EXAMPLES

```
  public interface Topic
{
   void understand();
}
class Topic1 implements Topic {
public void understand()
   {
      System.out.println("Got it");
   }
} class Topic2 implements Topic {
public void understand()
   {
      System.out.println("understand");
   }
} public class Subject {
public static void main(String[] args)
   {
      Topic t = new Topic1();
      t.understand();
   }
}
```

EXAMPLE

```
 class Volume
{
    public static void main(String args[])
    {
       Box b = new Box(5,5,5);
       System.out.println(b.getVolume());
    }
}
final class Box
{
    private int volume;
    Box(int length, int width, int height)
    {
       this.volume = length * width * height;
    }
    public int getVolume()
    {
        return volume;

    }
}
```

EXAMPLE

```
public class App {

 Job job;

 public App(Job job) {
  this.job = job;
 }

 public void display() {
  job.display();
 }

 public static void main(String args[]) {
  Engineer engineer = new Engineer();
  App app = new App(engineer);
  app.display();
 }

}
```

CONTROL STATEMENTS:

1: IF ESE STATEMENTS

EXAMPLES :

```
            class IfStatement {
  public static void main(String[] args) {

    int number = 10;

    // checks if number is greater than 0
    if (number > 0) {
      System.out.println("The number is positive.");
    }
else
{
    System.out.println("Statement outside if block");
  }
}
}
```

2: DO WHILE loops

EXAMPLES :

```
  class doWhileStatement {
  public static void main(String[] args) {
        int i = 0;
do {
  System.out.println(i);
  i++;
}
while (i < 5);
}
}
```

3: WHILE loops:

EXAMPLES :

```
class whileStatement {
  public static void main(String[] args) {
  int i = 0;
while (i < 5) {
  System.out.println(i);
  i++;
}
}
}
```

3 : FOR LOOPS:

EXAMPLES :

```
class forStatement {
  public static void main(String[] args) {
           for (int i = 0; i < 5; i++) {
  System.out.println(i);
}
}
}
```

4: SWITCH STATEMENTS

EXAMPLES :

```
class switchStatement {
  public static void main(String[] args) {
 int day = 4;
switch (day) {
  case 1:
    System.out.println("Monday");
    break;
  case 2:
    System.out.println("Tuesday");
    break;
  case 3:
    System.out.println("Wednesday");
    break;
  case 4:
    System.out.println("Thursday");
    break;
  case 5:
    System.out.println("Friday");
    break;
  case 6:
    System.out.println("Saturday");
    break;
  case 7:
    System.out.println("Sunday");
    break;
```

DATA TYPES :

```
class Main {
  public static void main(String[] args) {
 byte b =31;
short sh=331;
long l=3131313L;
int myNum = 5;
float myFloatNum = 5.99f;
double number = -42.3;
char myLetter = 'D';
boolean myBool = true;
String myText = "Hello";
System.out.println(b);
System.out.println(sh);
System.out.println(l);
System.out.println(mynum);
System.out.println(myFloatNum);
System.out.println(myBool);
System.out.println(myText);
}
}
```

CONSTRUCTORS:

EXAMPLES

```
        public class Student
{

    String name;
    String course;
    int age;

    public Student(String name, String course,int age)
    {
        this.name = name;
        this.course = course;
        this.age = age;
    }
    public String getName()
    {
        return name;
    }


    public static void main(String[] args)
    {
        Student s1 = new Student("MAHESH","CSE",23);

        System.out.println(s1.getName());
    }
```

ZERO PARAMETER CONSTRUCTOR AND PARAMETERISED CONSTRUCTOR :

EXAMPLES:

```
    import java.io.*;

class Student
{
    int num;
    int id;
    String name;
    student()
    {
        System.out.println("Constructor called");
    }
}
    student(String name, int id)
    {
        this.name = name;
        this.id = id;
    }
}

class School
{
    public static void main (String[] args)
    {
        student student1 = new Student();

        System.out.println(student1.name);
        System.out.println(student1.num);
        student student2 = new student ("adam", 1);
        System.out.println("studentName :" + student2.name +
                            " and student2 :" + student2.id);
    }
}
```