An overview of Java:- Java was developed by James Gosling, Patric naughton, chris worth, Ed Frank and mike sheridan at sunmicro system in 1991. This language was initially called "Oak" but was renamed "Java" in 1995.

Initially Java was not developed for internet. Instead, it can be used to create software to be embeded in various consumer electronic devices such as microwave ovens and remote controls.

The 'c' and 'c++' language can also be used in embedded software. If you written a program in c or c++ may require full 'c' or 'c++' compiles targetted for that CPU. The problem is that compiler are expensive and time consuming to create. In an attempt to find such a solution 'Gosling' and others begin to work on a platform independent language that could be used to produce the code that would run on variety of CPU's under different environments this effort ultimately leads to the creation of Java.

Features of Java language (Buzz words of Java)

1. Simple
2. Secure
3. Portable
4. Object oriented
5. Robust
6. Multithreaded
7. Architecture Neutral
8. Interpreted and High Performance
9. Distributed
10. Dynamic.

1. **Simple** : Java was designed to be easy for the professional Programmer to learn and use effectively. If you really understand the basic concepts of Object oriented Programming, it is very easy to learn.

2. **Secure** :- When we download a normal program you are risking a viral infection. When you use a java compatible web browser you can safely download java applets without fear of viral infection. Java achieves this protection by confining a Java Program to the Java execution environment and not allowing it to access the other parts of the computer.

Portability :- Java is portable means programs written in Java will work on different operating system and different machines.

Object-oriented :- since it is object oriented, we can write secure programs by using data encapsulation and reusability of code by inheritance concept.

Robust :- Java is Robust language for effective memory management and exceptional handling. In 'C' or 'C++' the programmer manually allocate and free all dynamic memory. This sometimes leads to problem, because programmers forget to free the memory that has been previously allocated. The Java eliminates these problems by managing memory allocation and de-allocation.
Exception condition in traditional environments often arises in situation such as division by zero or file not found and so on.
The Java handles these type of exceptions effectively at the time of runtime.

Multithreaded :- Java supports multithreaded programming which allows you to write programs that do many tasks simultaneously.

Architecture neutral :- one of the main problems facing by the programmers is that no guarantee exists that if you write a program today, it will run tomorrow even on the same machine. operating system upgrades, processor upgrades and changes in core system resources, can all combine to make a program malfunction. The Java designers made several hard decisions in the Java language and Java virtual machine in an attempt to alter this situation. Their goal was "write once, run anywhere, any time, forever".

Interpreted and high Performance :- Java enables the creation of cross platform programs by compiling to an intermediate representation called Java byte code. This code can be interpreted in any system that provide a java virtual Machine. Java was designed to perform well on very lower power CPU's.

Distributed:- Java is designed for the distributed environment of the internet because it handles the TCP/IP protocols. This allows us to execute the procedure remotely. This can be achieve by using the interface called RMI (Remote method Invocation).

Dynamic:- Java programs carry with them sustained amount of runtime information that is used to verify and resolves access to object at runtime. This make it possible to dynamically link code in a safe manner.

OOP's Concepts (Principles)

There are 3 OOP's Principles

1. Data Encapsulation
2. poly morphism
3. Inheritance.

1. Data Encapsulation :- The process of combining data and associated functions into a single entity called data encapsulation. Since data and function are combined into single, data can be accessed and processed only by the associated function.

Hence data is secure. Application of data encapsulation is 'class concept'.

2. Polymorphism :- Poly - many
morphism - forms.

It is an ability take more than one form.
The method overloading is an application of polymorphism.

3. Inheritance :- The mechanism of deriving new class from old class is called inheritance. The old class is called super class. the new class is called subclass.

Comments

Java supports three types of comments

1. Single line comments (//)
2. multiline comments (/* ... */)
3. Documentation comment (/** ... */)

This type of comment is used to produce an HTML file that documents your program automatically.

Data types (simple types)

The Java defines 8 simple types or data types. They are

- byte – 1 byte
- short – 2 bytes
- int – 4 bytes
- long – 8 bytes

(integers)

- float – 4 bytes
- double – 8 bytes

(float)

- char – 2 bytes
- boolean – (True or false)

Integers:- Java defined four integer type byte, short, int and long. All these are signed positive and negative values. Java does not support unsigned, positive only integers.

name	width
byte	1 byte (8 bits)
short	2 bytes (16 bits)
int	4 bytes (32 bits)
long	8 bytes (64 bits)

byte :- The smallest integer type is byte. This is signed 8-bit type that has a range from -128 to 127. Variables of type byte are especially useful when working with a stream of data from a network or file.

short :- short is a signed 16-bit type, it has a range from -32768 to 32767. This data type is used in Java. This type is most likely applicable to 16-bit computer.

int :- The most commonly used integer type is 'int'. It is signed 32-bit type. Variables of type int are commonly employed to control loops and to index arrays.

long :- long is signed 64-bit type and is useful for those occasion whose an int type is not large enough to hold the desired value. The range of long is quite large. This makes it useful when big whole numbers are needed.

Floating Point types :- Floating point numbers also known as real numbers, are used when you evaluating expressions that require fractional precision. There are 2 kinds of floating point types.

name	width
float	4 bytes (32-bit)
double	8 bytes (64-bit)

float :- the type float specifies a single precision value that uses 32-bit of storage. variables of type float are useful when you need a fractional component, but don't require a large degree of precision.

double :- This data type uses 64-bits to store a value. This data type is used when you need to maintain accuracy over many iterative calculation or manipulating large valued numbers.

characters :- In Java, the data types used to store characters is char. Java uses unicode to represent characters. uni-code defines fully international character set that can represent all of the characters found in all human languages. For exc:- latin, Greek, Arabic and so on.

For this purpose it requires 16 bits. Thus, in Java char is a 16-bit type. The range of char is 0 to 65,536. There are no negative values.

boolean:- Java has a simple type called boolean for logical values. it can have only one of the two possible values true or false.

Tokens

The smallest unit of a program is called a token.

The tokens in java are:

1 **keywords:-** These words have predefined meaning.

ex:- class, public, static, void, int - -

2 **Identifiers** :- Names used for classes, methods, variables.

Rules for Identifiers

1. must begin with letter, $ or _.
2. Cannot be a keyword
3. Blank spaces cannot be used.

ex:- sum, total_sal, $sum, _sum; valid Identifiers

Invalid: sum avg There is a blank space

int → a keyword

2sum → starts with a digit

literals

Constant values assigned to variables.

types :- Integel : 10, -20

floating type : 10.4, 5.0

Character : 'A'

String : "hello"

boolean : true, false.

Operators

Symbols that perform operations on variables and values.

Arithmetic : +, -, *, /, %

Relational : <, >, ==, >=, <=, !=

logical : !!, &&, !

Bitwise operators : !, &, !, ~, ^, <<, >>

Conditional :- ?, :

Separators (Delimiters) :- characters that help to define the structure of a Program.

Parenthesis (), Braces { }, Brackets : [ ], semicolon ;

Type conversion (implicit / widening conversion)

Happens automatically by the java compiler when you assign a value of a smaller datatype to a larger data type. it also called widening conversion.

it does not require a explicit code and no data loss.

```
int a = 10;
long b = a; // int to long, automatic type conversion
```

Type casting (Narrowing conversion)

Also called explicit casting, it is used when converting a larger data type to a smaller one.

```
double a = 5.8;
int b = (int) a; // double to int
```

it requires a explicit code and data may be lost.

operators precedence

() Parenthesis	— I	priority
*, /, %	— II	
+, −	— III	

Branching and looping statements

Simple If statement

- if and switch are branching statements

The general form of If statement is

```
If (condition)
  {
    statement-Block
  }
statement-x
```

The 'If' statement is used to make decisions.

The condition is tested, If it is true, statement-Block is executed. If the statement is false, the statement-Block will be skipped.

```
a = 5;
If (a > 2)
  {
    System.out.println("a is big");
  }
```

If-else

```
If (condition)
  {
    statement-1
  }
else
  {
    statement-2
  }
statement-x
```

first, the condition is tested, If it is true, statement-1 will be executed. otherwise, statement-2 will be executed.

```
If(a>b)
 {
    System.out.println("a is big");
 }
else
 {
    System.out.println("b is big");
 }
```

nested If

```
If (condition 1)
   {
      If(condition 2)
         {
            statement-Block 1
         }
      else
         {
            statement-Block 2
         }
   }
else
   {
      statement- Block 3
   }
```

If condition 1 and condition 2 are true then statement-block 1 will be executed. otherwise, statement-block 2 will be executed.

If condition 1 is false, then statement-block 3 will be executed.

```
if(n>0)
  {
    if(n%5==0)
     {
       system.out.println("+ve and divisible
                           by 5");
     }
    else
     {
       system.out.println("+ve but not
                           divisible by
                           5");
     }
  }
else
  {
    system.out.println("may be negative or
                        zero");
  }
```

else-if ladder

```
if(c1)
  {
    statement-block 1
  }
else if(c2)
    {
      statement-block 2
    }
  :
else
  { default statement } statement-x
```

In else-if ladder conditions are tested from top to downwards as soon as the true condition is found, then the associated statement-Blocks will be executed. After the execution, the control will be transfered to statement-x.

ex:-

```
if (n > 0)
  {
    System.out.println("positive number");
  }
else if (n < 0)
  {
    System.out.println("negative number");
  }
else
  {
    System.out.println("equal to zero");
  }
```

Switch statement

```
switch(expression)
 {
   case value 1:
                    statement-Block 1
                    break;
   case value 2:
                    statement-Block 2
                    break;
   case value 3:
                    statement-Block 2
                    break;
   default:
                    default block
 }
statement-x
```

The switch statement compares the value the expression with the case values. If the match found, then associated statement blocks are executed. The value of the expression does not match with any of the case values, default block will be executed.

```
switch(op)
{
    case 1: System.out.println("hello");
            break;
    case 2: System.out.println("hai");
            break;
    case 3; System.out.println("bye");
            break;
    default: System.out.println("invalid
                                option");
}
```

Repeatedly executing a block of code is called a loop. There are three types of loop.

1. while loop
2. do-while loop
3. for-loop

while loop

```
while (test condition)
  {
     body of the loop
  }
statement-x
```

The while is an entry-controlled loop statement. The test-condition is tested and if the condition is true, then the body of the loop is executed. After execution of the body, the test condition is once again re-evaluated and if it is true, the body is executed once again. This process of repeated execution of the body continues until the test-condition finally becomes false and the control is transferred out of the loop.

ex:-
```
i=1;
while(i<=4)
  {
    System.out.println(i);
  } i=i+1;
```

do-while loop

On some occasions it might be necessary to execute the body of the loop before the test is performed. Such situations can be handled with the help of the do statement. The general form is

```
do
  {
     body of the loop
  } while(test-condition);
```

On reaching the do statement, the program proceeds to execute the body of the loop first. At the end of the loop, the test-condition in the while statement is evaluated. If the condition is true, program continues to execute the body of the loop once again. This process continues as long as the condition is true. When the condition becomes false, the loop will be terminated and the control goes to the statement that appears immediately after the while statement.

```
i=1;
do
{
   System.out.println(i);
   i=i+1;
} while(i<=4);
```

for loop

The for loop is another entry-controlled loop that provides a more concise loop control structure. The general form of the for loop is

```
for( initialization ; test condition; increment or decrement )
{                1              2              4
        body of the loop
                 3
}
```

The execution of the for statement is as follows:

1. Initialization of loop control variable is done first.
2. The value of the loop control variable is tested using the test-condition. If the test condition is true, the body of the loop is executed; otherwise, the loop is terminated and the execution continues with the statement that immediately follows the loop.
3. When the body of the loop is executed, the control is transferred back to the for statement after evaluating the last statement in the loop. Now, the loop control variable is incremented or decremented and the new value of the control

variable is again tested to see whether it satisfies the test condition. If the condition is satisfied, the body of the loop is again executed. This process continues till the value of the loop control variable fails to satisfy the test condition.

ex:-
```
for(i=1; i<=4; i++)
{
    System.out.println(i);
}
```
it prints
1
2
3
4

break statement (Jumping Statement)

This statement is used to terminate a loop when there is a unusual condition occurs in the body of the loop.

ex:-
```
for(i=1; i<=10; i++)
{
    if(i==4)
    {
        break;
    }
    System.out.println(i);
}
```
it prints
1
2
3

labelled break statement

The general form of labelled break is

```
break label;
```

Here, label is the name of the label that identifies a block of codes. When this form of break executes, control is transferred out of the named block of code. The labelled block of code must enclosed the break statement.

ex:-

```
outer:
for(i=1; i<=3; i++)
{
    for(j=1; j<=10; j++)
    {
        if(j==4)
        {
            break outer;
        }
        System.out.println(i*j);
    }
}
```

Continue statement (Jumping statement)

Continue statement causes the loop to be continued with the next iteration after skipping any statements in between.

for ex:-

```
for(i=1; i<=10; i++)
{
    if(i%2==0)
    {
        continue;
    }
    System.out.println(i);
}
```

it prints

1
3
5
7
9

labelled continue

The general form of labelled continue is

```
continue label
```

In labelled continue statement, we give a label to a loop. when this continue statement is encountered with the label of the loop, it skips the execution of any statement within the loop for the current iteration and continues with the next iteration and condition checking in the labelled loop.

ex:-

```
outer:
for(i=1; i<=3; i++)
{
    for(j=1; j<=5; j++)
    {
        if(j%2==0)
        {
            continue outer;
        }
        System.out.println(i*j);
    }
}
```

Arrays

An array is a group of similar data items that are shared a common name.

Declaration of Arrays

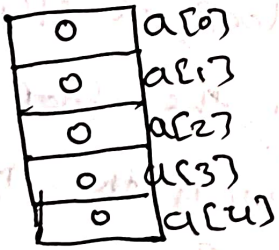
```
datatype arrayname[]
```

datatype can be any of the simple datatypes.
arrayname is the name used to store the values.

ex:- int a[];

a = new int[5];

The computer reserves five memory locations for a and all memory locations are initialized to zero.

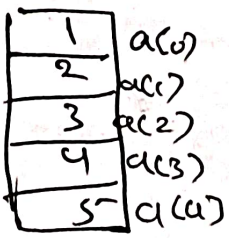


The above declaration of an array can be done in single line.

int a[] = new int[5];

Array initialization

int a[] = {1, 2, 3, 4, 5}



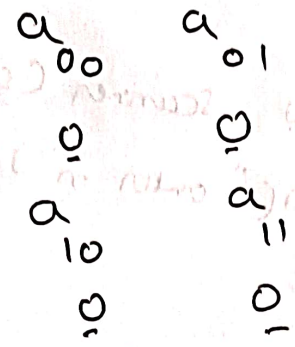
ex:- Program to read and print an array.

```
import java.util.*;
class Arraydemo
{
    public static void main(String args[])
    {
        int n, i;
        Scanner s = new Scanner(System.in);
        System.out.println("enter n");
        n = s.nextInt();
        int a[] = new int[n];
        System.out.println("enter array elements");
        for(i=0; i<n; i++)
        {
            a[i] = s.nextInt();
        }
        System.out.println("entered elements are");
        for(i=0; i<n; i++)
        {
            System.out.println(a[i]);
        }
    }
}
```

Two dimensional arrays

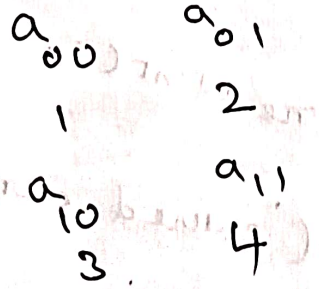
int a[][] = new int[2][2];

it will allocates two rows and two columns.



initialization of two dimensional arrays.

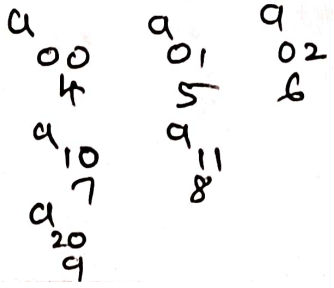
int a[][] = {{1,2},{3,4}};



Jagged Arrays

A it contains rows of variable length elements

for ex:-



for-each (enhanced for loop)

for-each is another array traversing technique without using an explicit index. The general form is

```
for(type var : collection
                or
               array)
{
    body of the loop
}
```

Instead of declaring and initializing a loop counter variable, you declare a variable that is same type as the base type of the array, followed by a colon, which is then followed by the array name.

In the body of the loop, you can use loop variable

ex:-

```
class foreachdemo
{
    Public static void main(String args[])
    {
        int a[] = {1,2,3,4};
        for(int i : a)
        {
            System.out.println(i);
        }
    }
}
```

it prints

1
2
3
4

classes and objects

class :- It is a mechanism allows us to combine data and operations on those data into a single unit.

The general form of declaring a class is as follows.

```
class classname
 {
    datatype  instance variable-1;
    datatype  instance variable-2;
        - -
        -
    datatype  instance variable-n;
    datatype  method name1(argument list)
       {
          = =
       }
    datatype method name2(argument list)
       {
          = =
       }
 }
```

The data, or variables, defined within a class are called instance variable. The code is contained with in the methods. Collectively, the methods and variables defined with in a class are called members of the class.

The instance variable are acted upon and accessed by the methods defined for that class. variables defined within a class are called instance variables because each instance of class (object) contains its own copy of the variables. Thus the data for one object is separate and unique from the data for others.

Object :- It is an instance of a class.

Program to add two numbers using class and object.

```
import java.util.*;
class Add
{
    int a;
    int b;
    int c;
    void getdata()
    {
        Scanner s = new Scanner(System.in);
        System.out.println("enter a, b");
        a = s.nextInt();
        b = s.nextInt();
    }
```

```
    void compute()
    {
        c = a + b;
        System.out.println("c=" + c);
    }
}
class sum
{
    public static void main(String args[])
    {
        Add x = new Add();
        x.getdata();   // x is an object.
        x.compute();
    }
}
```

As soon as we create an object the instance variables are initialized with default values.

for ex:- The object x contains a, b, and c are all initialized with zero.

x
a 0
b 0
c 0

when applied the getdata() method on x the user can give his own data.

The compute method perform operation on a and b. The result will be stored in c.

Passing data to a method from main program.

```
class sum
 {
    int a;
    int b;
    int c;
    void getdata(int m, int n)
     {
        a=m;
        b=n;
     }
    void compute()
     {
        c=a+b;
      System.out.println("c="+c);
     }
 }
class Dmethod
 {
    public static void main(String args[])
     {
        Sum s=new sum();
        s.getdata(10,20);
        s.Compute();
     }
 }
```

Returning a data from a method to main program

```
class sum
 {
    int a;
    int b;
    int c;
    void getdata(int m, int n)
     {
        a = m;
        b = n;
     }
    int compute()
     {
        c = a + b;
        return(c);
     }
 }
class Rdata
 {
    public static void main(String args[])
     {
        sum s = new sum();
        s.getdata(10,20);
        int s1 = s.compute();
        System.out.println(s1);
     }
 }
```

Pass object as a method parameter

```
class objpas
 {
    int a;
    int b;
    void getdata()
     {
        a = 10;
        b = 20;
     }
    void copy(objpas t)
     {
        a = t.a;
        b = t.b;
     }
    void printdata()
     {
        System.out.println(a);
        System.out.println(b);
     }
 }
 class objpasdemo
  {
    public static void main(String args[])
     {
        objpas ob = new objpas();
```

```
        ob.getdata();
        objpas ob1 = new objpas();
        ob1.copy(ob);
        ob1.printdata();
    }
}
```

Program to return an object from a method.

```
import java.util.*;
class complex
{
    double real;
    double imag;
    void getdata()
    {
        Scanner s = new Scanner(System.in);
        System.out.println("real and imaginary
                            part");
        real = s.nextDouble();
        imag = s.nextDouble();
    }
    void Putdata()
    {
        System.out.println(real + "i" + imag);
    }
```

```
Complex sum(complex c2)
  {
    complex t = new complex();
    t.real = real + c2.real;
    t.imag = imag + c2.imag;
    return(t);
  }
}
class complexdemo
  {
    Public static void main(String args[])
      {
        complex c1 = new complex();
        complex c2 = new complex();
        Complex c3 = new complex();
        System.out.println("enter first complex
                            number");
        c1.getdata();
        system.out.println("enter second complex
                            number");
        c2.getdata();
        c3 = c1.sum(c2);
        c1.putdata();
        c2.putdata();
        c3.putdata();
      }
}
```

method overloading (compile time polymorphism)

It is possible to define two or more methods with in the same class that share the same name with different argument list. when this is the case, the methods are said to be overloaded, and the process is referred to as method overloading.

```
class geometry
{
    void area(double r)
    {
        double a = 3.142*r*r;
        System.out.println("area of a
                            circle="+a);
    }
    void area(double b, double h)
    {
        double a = 0.5 * b * h;
        System.out.println("area of a
                            triangle="+a);
    }
}
```

```
class Moverload
{
  public static void main(String args[])
  {
    Geometry g = new Geometry();
    g.area(2);
    g.area(3,5);
  }
}
```

Constructor

- Constructor is a special method whose task is to initialize the object of a class.
- It is invoked automatically when the object of a class is created.
- It's name is same as the class name.
- It may contains parameter but it will not return any values.

Types of Constructor

1. Default constructor
2. Parameterized Constructor
3. Copy Constructor.
4. No-argument Constructor.

1. Default constructor :- A constructor with no palameters.

Note :- If constructor is not there in a class, automatically default constructor will be called and initialize the instance variables.

// Program to demonstrate default constructor

```
class Dcons
 {
    int a;
    int b;
    Dcons()
     {
       a = 10;
       b = 20;
     }
    void putdata()
     {
       System.out.println(a);
       System.out.println(b);
     }
}
class Dconsdemo
 {
    public static void main(String args[])
     {
       Dcons d = new Dcons();
} } d.putdata();
```

Palameterized constructor

Constructors which contain palameters are called palameterized constructors.

```
class PCons
 {
    int a;
    int b;
    PCons(int m, int n)
     {
       a = m;
       b = n;
     }
    void putdata()
     {
       System.out.println(a);
       System.out.println(b);
     }
 }
class PConsdemo
 {
    Public static void main(String args[])
     {
        PCons  P = new PCons(10,20);
        P.putdata();
     }
 }
```

(3) Copy constructor :- Used to create a new object by copying the fields of another object.

```
class ccons
 {
    int a;
    int b;
     ccons()
      {
         a = 10;
         b = 20;
      }
     ccons(ccons c)
      {
         a = c.a;
         b = c.b;
      }
     void putdata()
      {
         System.out.println(a);
         System.out.println(b);
      }
 }
class cconsdemo
 {
    public static void main(String args[])
      {
         ccons c1 = new cons()
         ccons c2 = new cons(c1);
      }
```

c1.putdata();
c2.putdata();

NO-Argument constructor

- This is a user-defined constructor that takes no arguments.

```
class Reva
 {
    Reva()
    {
      System.out.println("welcome to Reva");
    }
 }
class NArgdemo
 {
    public static void main(String args[])
      {
        Reva r = new Reva();
      }
 }
```

NOTE:- A default constructor will be called even though a constructor is not specified in a program.

A NO-Argument constructor will be called only when the user must be specified in the class.

Constructor overloading

multiple constructors in a class with different parameter lists.

```
class Coverload
 {
   String srn;
   String name;
   int marks;
   Coverload()
    {
      srn = "unknown";
      name = "unknown";
      marks = 0;
    }
   Coverload(String s, String n, int m)
    {
      srn = s;
      name = n;
      marks = m;
    }
   void printdata()
    {
      System.out.println("srn = " + srn + "name = " +
                         name + "marks = " +
                         marks);
    }
}
```

```
class coverloaddemo
{
  Public static void main(String args[])
  {
    Coverload c1 = new coverload();
    Coverload c2 = new coverload("XYZ",
                   "24MCA", "50");
    c1.printdata();
    c2.printdata();
  }
}
```

Call by value and call by reference

In Java call by value is achieved using primitive data types. when you pass a primitive datatype to a method, then it is called call by value.

when you pass an object to a method, then it called call by reference.

// Program to demonstrate call by value. (9)

```
class cvalue
 {
    void change(int m, int n)
     {
        m = m+10;
        n = n+10;
     }
 }
class cvaluedemo
 {
    public static void main(String args[])
     {   int a=10, b=20;
         cvalue c = new cvalue();
         c.change(a, b);
         System.out.println("a=" + a + "b=" + b);
     }
 }
```

Note:-

Any changes make to formal arguments that will not affect to the actual arguments.

Call by reference

```
// Program to demonstrate call by reference

class cref
{
    int a;
    int b;
    cref()
    {
        a=10;
        b=20;
    }
    void change(cref c)
    {
        c.a = c.a + 10;
        c.b = c.b + 20;
    }
    void printdata()
    {
        system.out.println("a=" + a + "b=" + b)
    }
}
class crefdemo
{
    public static void main(String args[])
    {
        cref cr = new cref();
        cr.change(cr);
        cr.printdata();
    }
}
```