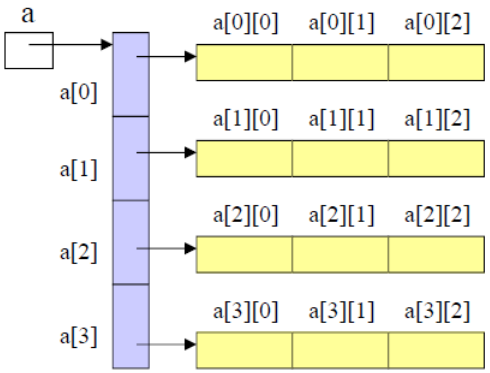
Classes and Objects

Multidimensional Arrays

- 2-dimensional arrays == matrix

	0	1	2
0	a[0][0]	a[0][1]	a[0][2]
1	a[1][0]	a[1][1]	a[1][2]
2	a[2][0]	a[2][1]	a[2][2]
3	a[3][0]	a[3][1]	a[3][2]

- In Java: arrays of arrays



Declaration and instantiation

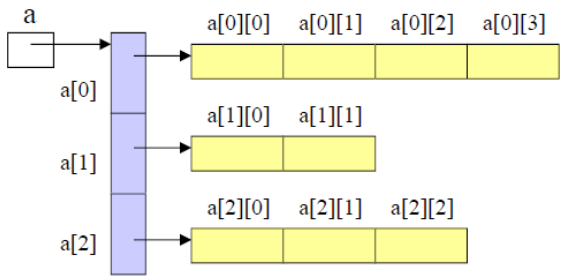
```
int[][] a;
a = new int[4][3];
```

Access

```
a[i][j] = a[i][j+1];
```

Multidimensional Arrays

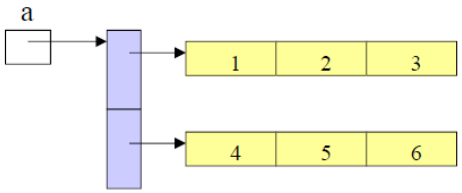
- Rows can be of arbitrary length




```
int[][] a = new int[3][];
a[0] = new int[4];
a[1] = new int[2];
a[2] = new int[3];
```

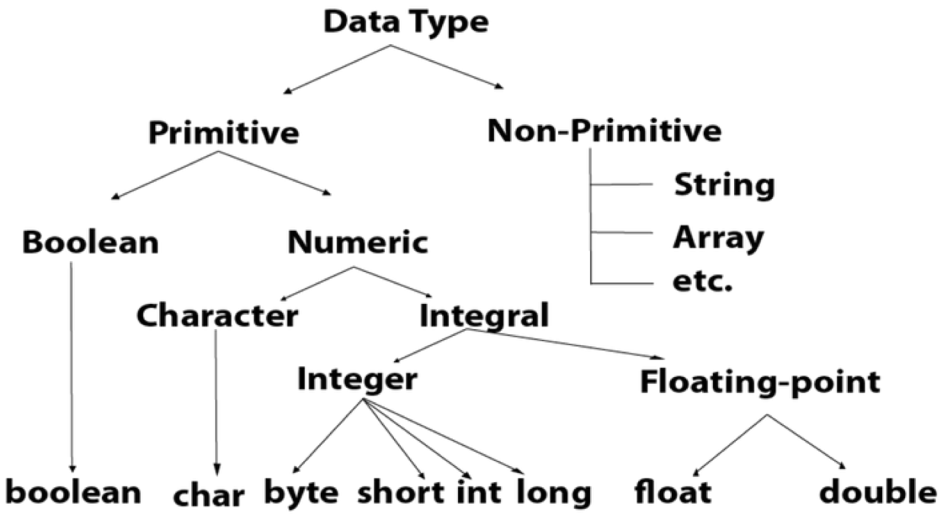
- Initialisation

```
int[][] a = {{1, 2, 3},{4, 5, 6}};
```



Looking back ..

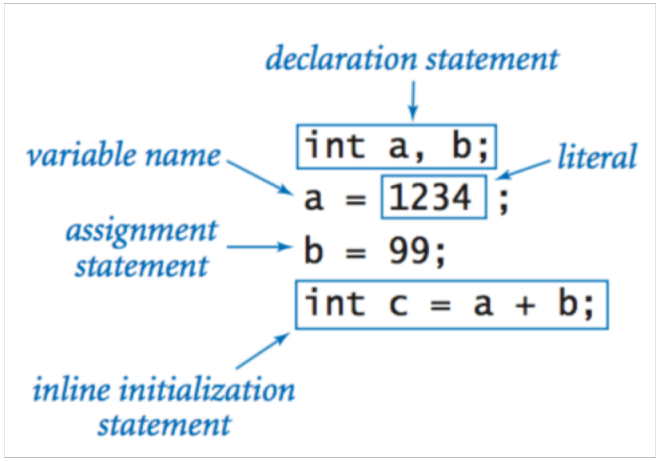
- Scalar data types
  - „basic data types“ int, byte, short, int, long, float, double, boolean, char
  - Variable contains value
- Aggregated data types
  - More than a single basic data organized through a single name
  - cp. arrays ...

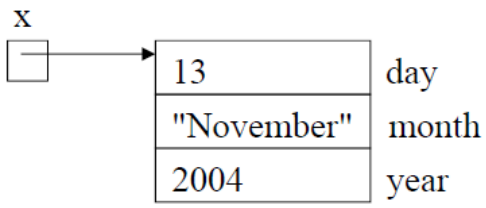


Looking back ...

- Reference data type
  - variable stores reference / address
  - not a value

- In Java
  - basic data types -> by value
  - everything else -> by reference

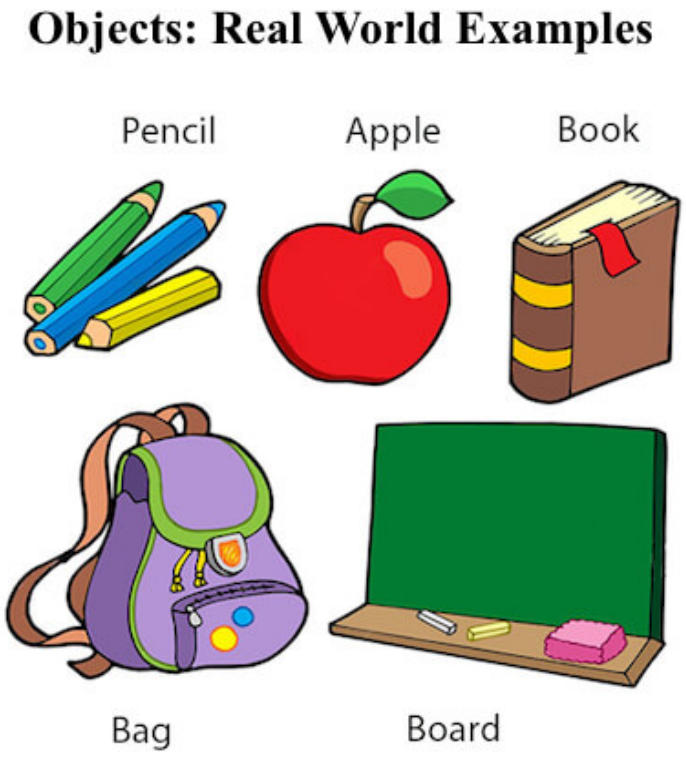




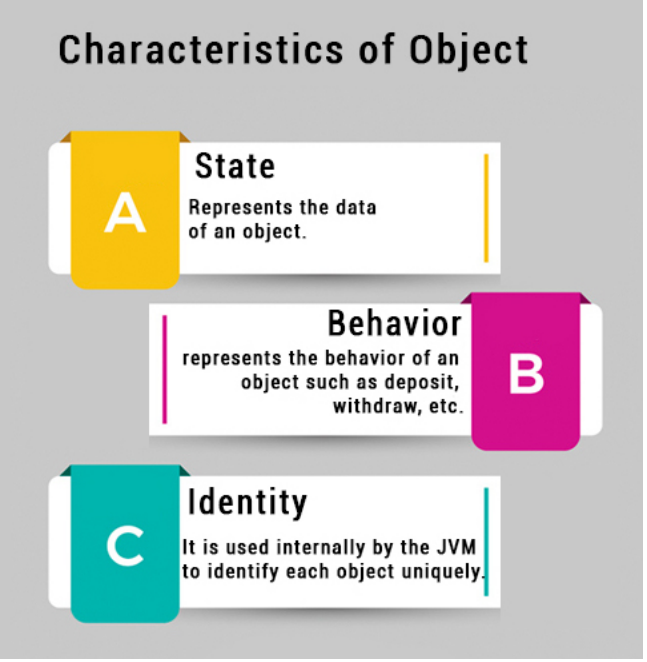

WARNING
Falling objects

What is actually an object?

- I mean really?



What is an object in Java?



- An entity that has state and behavior
  - e.g. chair, bike, marker, pen, table, car
- An object has three characteristics
  - **State:** represents the data (value) of an object.
  - **Behavior:** represents the behavior (functionality) of an object such as deposit, withdraw, etc.
  - **Identity:** An object identity is typically implemented via a unique ID. The value of the ID is not visible to the external user. However, it is used internally by the JVM to identify each object uniquely.
  - E.g., Pen is an object. Its name is Reynolds; color is white, known as its state. It is used to write, so writing is its behavior.
- An object is an instance of a class. A class is a template or blueprint from which objects are created. So, an object is the instance(result) of a class.

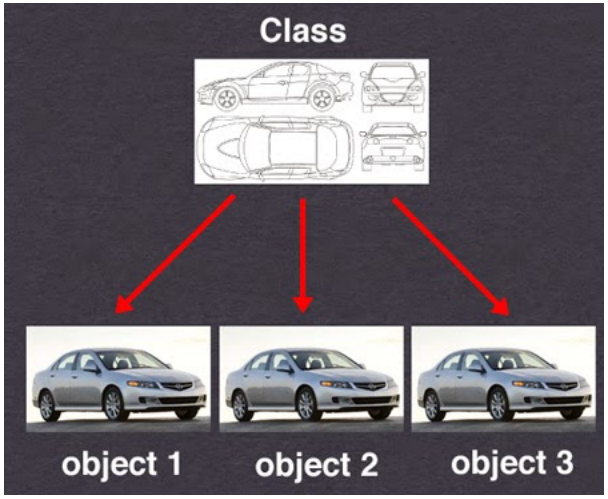
What is a class in Java?

- A class is a group of objects which have common properties. It is a template or blueprint from which objects are created. It is a logical entity. It can't be physical.
- A class can contain: fields, methods, constructors, blocks, nested class and interface
- Syntax to declare a class:

```
class <class_name>{
    field;
    method;
}
```

- Classes have instance variables
  - variable which is created inside the class but outside the method
    - Does not allocate memory at compile time but at run time when instance is created
- Method in the class defines the behavior
  - Code reusability, code optimization
- New keyword is used to allocate memory at runtime.

Objects and classes



- Class is like a template
  - from which instances (objects) are created

- Objects (instances) of a class have to be created explicitely before use.
  - Variable otherwise have the value `null`

```
Car myCar; -> null
Car myCar = new Car(); -> car with inital values
```

Declaration of Classes

Single file

```
class C1 {
   ...
}
class C2 {
   ...
}
class MainProgram {
  public static void
         main (String[] arg) {
         ...
   }
}
```

MainProgram.java

Compile

$> javac MainProgram.java

Multiple files

```
class C1 {
   ...
}
class C2 {
   ...
}
class MainProgram {
  public static void
         main (String[] arg) {
         ...
   }
}
```

C1.java

C2.java

MainProgram.java

Compile

$> javac MainProgram.java C1.java C2.java

Java classes example

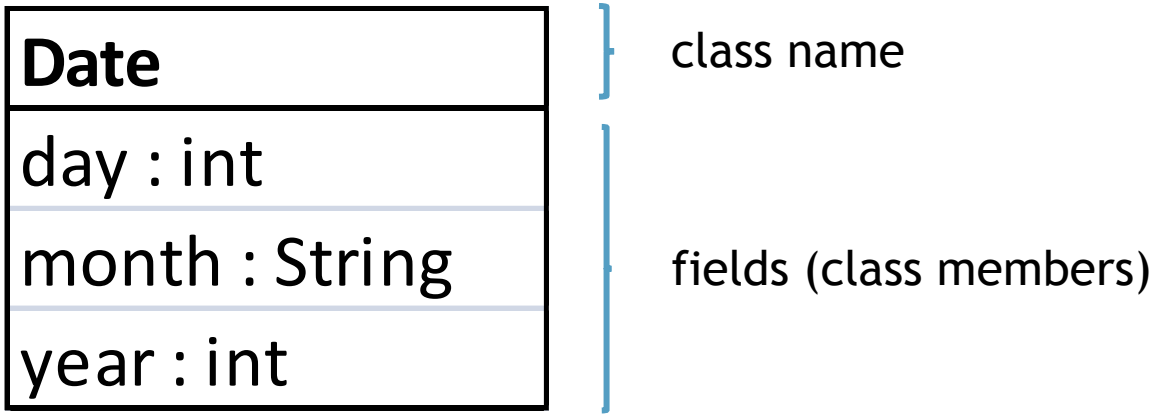
- Example: Store a data in a single structure.
  - day, month, year, ...

- Basic data types not practical for this ...
  - storing more than one
  - return values of functions
  - comparing to other dates

Date
day : int
month : String
year : int

Java classes example

- Combine all necessary variables in one structure:



Data Type Class

- Declaration

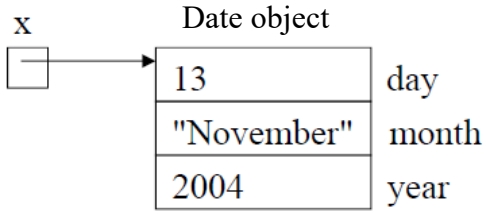
- Data type usage
- Access

```
class Date {
   int day;
   String month;
   int year;
}
```

fields of class date

```
Date x, y;
```

```
x.day = 13;
x.month = "November";
x.year = 2004;
```



Date variables are references / addresses to objects.

Object and class example code

```
//Creating Student class.
class Student{
 int id;
 String name;
}
//Creating another class TestStudent1 which contains the main method
class TestStudent1{
 public static void main(String args[]){
  Student s1=new Student();
  System.out.println(s1.id);
  System.out.println(s1.name);
 }
}
```

3 Ways to initialize objects

Initializing an object means storing data into the object.

By reference variable

By method

By constructor (more details later on this)

Initialization through reference

```
class Student{
 int id;
 String name;
}
class TestStudent2{
 public static void main(String args[]){
  Student s1=new Student();
  s1.id=101;
  s1.name="Sonoo";
  System.out.println(s1.id+" "+s1.name);//printing members with a white space
 }
}
```

We can also create multiple objects and store information in it through reference variable.

```
class Student{
 int id;
 String name;
}
class TestStudent3{
 public static void main(String args[]){
  //Creating objects
  Student s1=new Student();
  Student s2=new Student();
  //Initializing objects
  s1.id=101;
  s1.name="Sonoo";
  s2.id=102;
  s2.name="Amit";
  //Printing data
  System.out.println(s1.id+" "+s1.name);
  System.out.println(s2.id+" "+s2.name);
 }
}
```

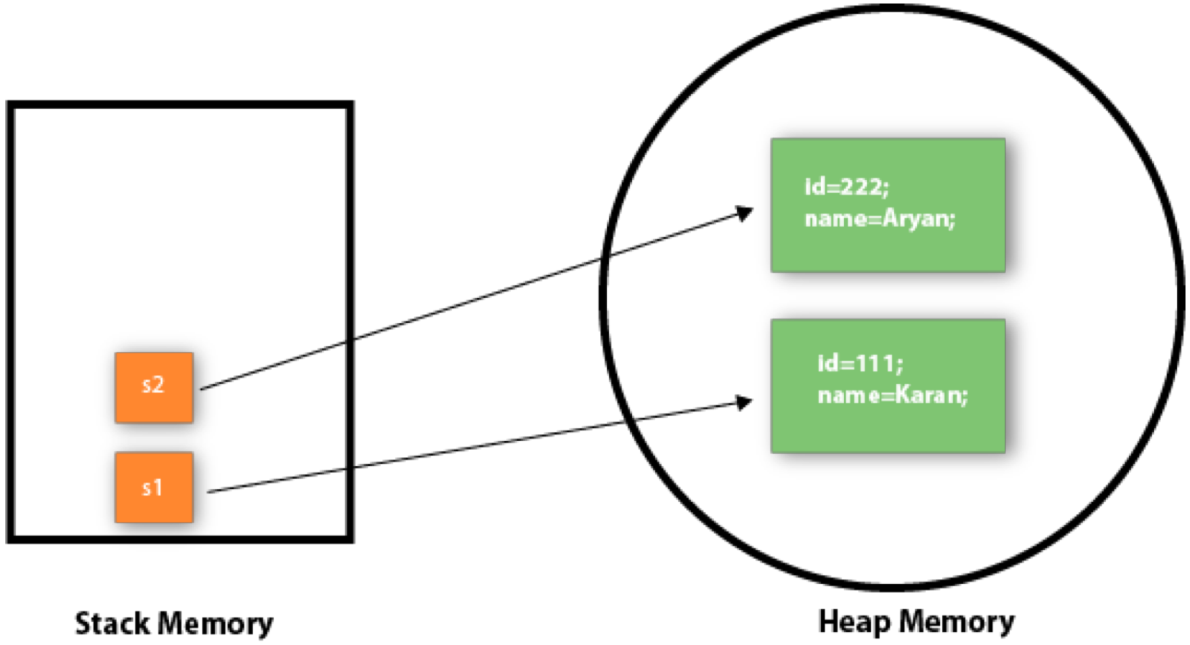
Initialization through method

- Initializing the value to these objects by invoking the insertRecord method

```
class Student{
 int rollno;
 String name;
 void insertRecord(int r, String n){
  rollno=r;
  name=n;
 }
 void displayInformation(){System.out.println(rollno+" "+name);}
}
class TestStudent4{
 public static void main(String args[]){
  Student s1=new Student();
  Student s2=new Student();
  s1.insertRecord(111,"Karan");
  s2.insertRecord(222,"Aryan");
  s1.displayInformation();
  s2.displayInformation();
 }
}
```

Memory?

- Objects get the memory in heap memory area.
- Reference variables refer to the object allocated in the heap memory area.



Objects intialisation and referencing

```
Date x, y;
```

reserves memory for the address

x,y have value null

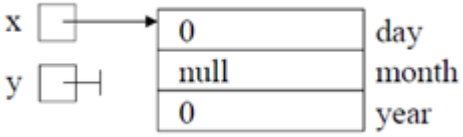
Instantiation

```
x = new Date();
```

creates a new Date object and assigns ist address to x.

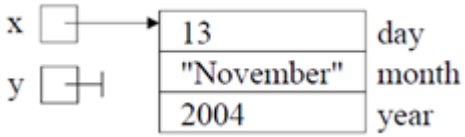
Initial values are

0, null, false or ‚'\u0000'



Usage

```
x.day = 13;
x.month = „November“
x.year = 2004;
```



Assignments

Reference / address assignment

changes x.day too!

Assignments

```
class Date {
   int day;
   String month;
   int year;
}
```

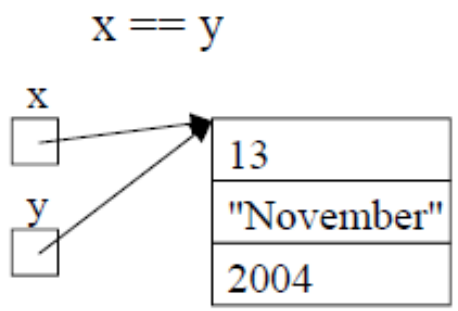
```
class Address {
   int number;
   String street;
   int zipCode;
}
```

```
Date d1, d2 = new Date();
Address a1, a2 = new Address();
```

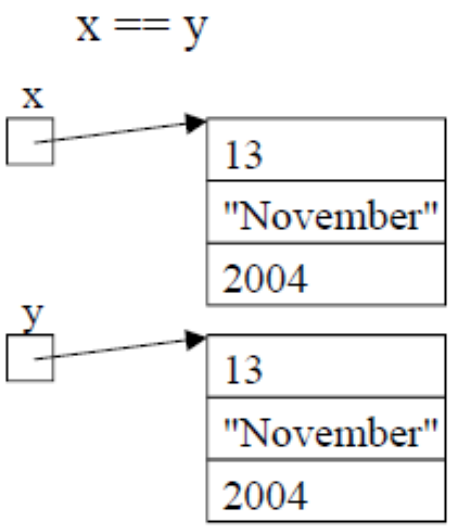
```
d1 = d2;   // ok, same type
a1 = a2;   // ok, same type
d1 = a2;   // not ok, different type (although structure is the same)
```

Comparing references

- `x == y` und `x != y` ... compares references
- `<, <=, >, >=` ... not applicable



x == y returns true



x == y returns false

Compares actual values

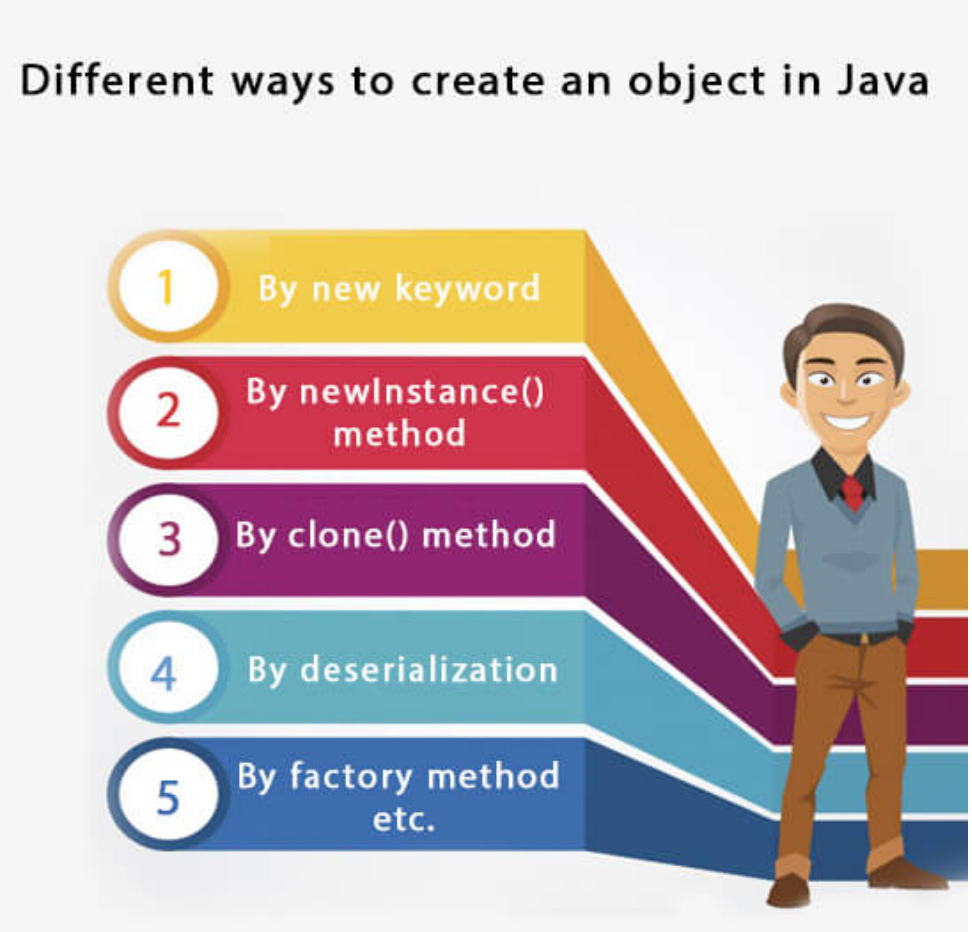
- Has to be implemented by a method.

```
static boolean equalDate (Date x, Date y) {
  return x.day == y.day &&
    x.month.equals(y.month) &&
    x.year == y.year;
}
```

Employee class example

```
class Employee{
    int id;
    String name;
    float salary;
    void insert(int i, String n, float s) {
        id=i;
        name=n;
        salary=s;
    }
    void display(){System.out.println(id+" "+name+" "+salary);}
}
public class TestEmployee {
public static void main(String[] args) {
    Employee e1=new Employee();
    Employee e2=new Employee();
    Employee e3=new Employee();
    e1.insert(101,"ajeet",45000);
    e2.insert(102,"irfan",25000);
    e3.insert(103,"nakul",55000);
    e1.display();
    e2.display();
    e3.display();
}
}
```

What are the different ways to create an object in Java?



- We will learn these ways later…

Anonymous object

- Anonymous simply means nameless.
  - An object which has no reference is known as an anonymous object.
  - It can be used at the time of object creation only.
  - If you have to use an object only once then it might be a clever joice…
  - Usually you would use myObject = new Object() but…

```
class Calculation{
 void fact(int  n){
  int fact=1;
  for(int i=1;i<=n;i++){
   fact=fact*i;
  }
 System.out.println("factorial is "+fact);
}
public static void main(String args[]){
 new Calculation().fact(5);//calling method with anonymous object
}
}
```

Multiple objects by one type only

- We can create multiple objects by one type only as we do in case of primitives.
- Initialization of primitive variables:

int a=10, b=20;

//creating two primitive datatypes

- Initialization of reference variables:

Rectangle r1=**new** Rectangle(), r2=**new** Rectangle();

//creating two objects

What can we do with classes and objects?

- Classes can use other classes
  - and extend on that

```
class Point {
  int x, y;
}

class Polygon {
  Point[] pt;
  int color;
}
```

Has a relation – also called aggregation, later more

What can we do with classes and objects?

- Implement methods with multiple return values

```
class Time {
   int hours, minutes, seconds;
}
class Program {
   static Time convert (int seconds) {
      Time myTime = new Time();
      myTime.hours = seconds / 3600;
      myTime.minutes = (seconds % 3600) / 60;
      myTime.seconds = seconds % 60;
      return time;
   }
   public static void main (String[] arg) {
      Time myTime = convert(10000);
      System.out.println(tmyTime.hour + ":" + myTime.minute + ":" + myTime.seconds);
   }
}
```

What can we do with classes and objects?

- Combination of classes and arrays

```
class Person {
   String name, phoneNumber;
}

class Phonebook {
   Person[] entries;
}

class Program {
   public static void main (String[] arg) {
      Phonebook phonebook = new Phonebook();
      phonebook.entries = new Person[10];
      phonebook.entries[0].name = “John Doe"
      phonebook.entries[0].phoneNumber = "+47 123 456 789"
      // ...
   }
}
```

What can we do with classes and objects?

- Combination of classes and arrays

```
class Person {
   String name, phoneNumber;
}

class Phonebook {
   Person[] entries;
}

class Program {
   public static void main (String[] arg) {
      Phonebook phonebook = new Phonebook();
      phonebook.entries = new Person[10];
      phonebook.entries[0].name = “John Doe"
      phonebook.entries[0].phoneNumber = "+47 123 456 789"
      // ...
   }
}
```

Hands-on on classes and objects in class

- Coding!
- Real world example
- Banking system that allows deposit and withdraw money from account

- Sneak peak. Focus even more on coding:
- Constructors
- Static key word in Java
- This key word in Java
- Inheritance in Java
- Aggregation in Java
- Method overloading and overriding in context with OOP
- … interfaces, polymorphism, …