

প্রথম খণ্ড

# মাতৃভাষায় জাভা শিখি

আব্দুজ জামি



নিহারন প্রকাশনী

# মাতৃভাষায় জাভা শিখি

আব্দুজ জামি

প্রথম খণ্ড

পিডিএফ সংস্করণ - ১.০

মাতৃভাষায় জাভা শিখি (১ম খণ্ড)

গ্রন্থ সত্ত্ব : আব্দুজ জামি

প্রথম প্রকাশ : অক্টোবর ২০২১

মূল্য :

পিডিএফ সংস্করণ : ২০ টাকা মাত্র

# ভূমিকা

বর্তমান দুনিয়ায় কম্পিউটার ব্যবহার করেনা এমন মানুষ খুঁজে পাওয়া বেশ কঠিন। ছোট একটি মোবাইল ফোন সেটিও একটি কম্পিউটার। আবার একটি ক্যালকুলেটর ও একটি কম্পিউটার। তাই আজ শিক্ষিত-অশিক্ষিত, ব্যবসায়ী-চাকুরীজীবী সবাই কম্পিউটার ব্যবহার করে। আমরা যখন কম্পিউটার এ কোন কাজ করি তখন কিন্তু ওই কাজটা খুব সহজেই করে ফেলি। কিন্তু আমরা যখন ভাবি যে কম্পিউটার কে দিয়ে এই কাজ গুলো করানো হয় কিভাবে তখন আমাদের মাথা ঘুরে যায়। মনে হয় না জানি কত কঠিন জিনিস। মূলত কাজ টা কঠিন হলেও ততটাও কঠিন হয়। একটু চেষ্টা করলেই আমরাও কম্পিউটার কে দিয়ে আমাদের ইচ্ছামত কাজ করিয়ে নিতে পারি। কম্পিউটার কে দিয়ে কাজ করানোটাই হল প্রোগ্রামিং। প্রোগ্রামিং করার জন্য অনেক ধরনের ভাষা আছে। মানুষের ভাষা তো আর কম্পিউটার বুঝে না। আমরা অনেকেই হয়ত জানি যে কম্পিউটার সহ যেকোনো ইলেক্ট্রিক সার্কিট কাজ করে এক এবং শূন্য এই দুইটা সংকেত এর মাধ্যমে। এক এবং শূন্য যথাক্রমে চালু এবং বন্ধ বুঝায়। এক আর শূন্য দিয়ে লেখা সংকেত আবার মানুষের পক্ষে বোঝা কষ্টকর। তাই কিছু প্রোগ্রামিং ল্যাঙ্গুয়েজ আছে যা মানুষ সহজেই বুঝতে পারে। এই ভাষায় প্রোগ্রাম লেখার পর টা একটি প্রক্রিয়ায় মেশিনের বোধগম্য ভাষায় পরিবর্তিত হয়। এরকম প্রক্রিয়া গুলোর মধ্যে উদাহরণস্বরূপ কম্পাইলার এবং ইন্টারপ্রেটর এর নাম বলা যায়। জাভাও এরকমই একটি ল্যাঙ্গুয়েজ। এই বইটিতে আমি খুব সহজ ভাষায় অল্প কথার মধ্যে জাভা ল্যাঙ্গুয়েজ এর নানান বিষয় নিয়ে কথা বলেছি। আশা করি বইটি পড়ে আপনারা উপকৃত হবেন।

# লেখক পরিচিতি

লেখকের নাম আব্দুজ্জামি। জন্ম ২০০১ সালের ২৮ জুন চট্টগ্রামের হাটহাজারি উপজেলায়। পৈত্রিক নিবাস মানিকগঞ্জ জেলার চান্দরা গ্রামে। পিতা মোঃ হযরত আলী সরকারি চাকুরীজীবী। মাতা মনোয়ারা বেগম একজন গৃহিণী। লেখক এর একটি ছোট ভাই আছে নাম আব্দুস সামি। বাবার চাকুরির সুবাদে বাংলাদেশের বেশ অনেক জায়গায় থাকতে হয়েছে তাকে। স্কুল ও বদলেছেন বার বার। চট্টগ্রামের ইকরা ইন্সটিটিউট এ লেখাপড়া শুরু। এর পর নেত্রকোনার লার্নিং পয়েন্ট কিন্ডারগার্টেন, নারায়ণগঞ্জের সদাসদি বহুমুখি উচ্চ বিদ্যালয়, কুমিল্লার লাকসাম উচ্চ বালক বিদ্যালয়, গাজীপুরের রানী বিলাশমনি সরকারি বালক উচ্চ বিদ্যালয় এবং নটর ডেম কলেজে পড়ালেখা করেছেন। বর্তমানে রাজশাহী প্রকৌশল ও প্রযুক্তি বিশ্ববিদ্যালয়ে কম্পিউটার সাইন্স এবং ইঞ্জিনিয়ারিং বিভাগে অধ্যয়নরত আছেন। তার বানানো দুইটি মোবাইল অ্যাপ প্লে স্টোরে বেশ জনপ্রিয়। তার একটি হচ্ছে Niharon Class Manager এবং অন্যটি Niharon Shop Manager.

# লেখকের কথা

এটি আমার প্রকাশ করা প্রথম বই। বইয়ে ভুল ত্রুটি থাকতে পারে। ভুল ত্রুটি গুলো ধরিয়ে দিতে আমাকে মেইল করতে পারেন। আমাকে মেইল করার ঠিকানা [abduz.zami@gmail.com](mailto:abduz.zami@gmail.com)। বইটির পিডিএফ এর শুভেচ্ছা মূল্য মাত্র ২০ টাকা। তবে কারো মূল্য প্রদানে সমস্যা থাকলে সে নির্বিধায় বিনামূল্যে বইটি পরতে পারে। বইটি সম্পর্কে আপনার মতামত জানাতে আমাকে মেইল করতে পারেন। আপনাদের সাড়া পেলে খুব শিগ্ৰই বইটির হার্ড কপি আপনাদের সামনে নিয়ে আসব। বইটির পিডিএফ এই লিঙ্কে পাওয়া যাচ্ছে <https://sites.google.com/view/niharonprokashoni/home>।

# সুচিপত্র

জাভা কি কেন শিখব  
 অবজেক্ট ওরিয়েন্টেড প্রোগ্রামিং  
 জাভা ফাইলের গঠন  
 প্রিন্ট করা  
 ডাটা টাইপস  
 ইউজার থেকে ইনপুট নেওয়া  
 ফরমেট স্পেসিফায়ার  
 অপারেটর  
 কন্ডিশনাল লজিক  
 লুপ  
 অ্যারে  
 মেথড  
 স্ট্রিং  
 জাভা প্রজেক্ট এর গঠন  
 প্যাকেজ  
 ক্লাস এবং অবজেক্ট  
 রেফারেন্স টাইপের মেমোরি বন্টন  
 কম্পট্রাক্টর  
 প্রিমিটিভ টাইপ বনাম রেফারেন্স টাইপ  
 কল বাই ভ্যালু  
 কল বাই রেফারেন্স  
 রিপার ক্লাস  
 ইনহেরিটেন্স  
 অ্যাক্সেস মডিফায়ারস ও এনকেপ্সুলেশন

গেটার এবং সেটার মেথড

মেথড ওভাররাইডিং

মেথড ওভারলোডিং

কমান্ড লাইন আর্গুমেন্ট



# জাভা কি কেন শিখব

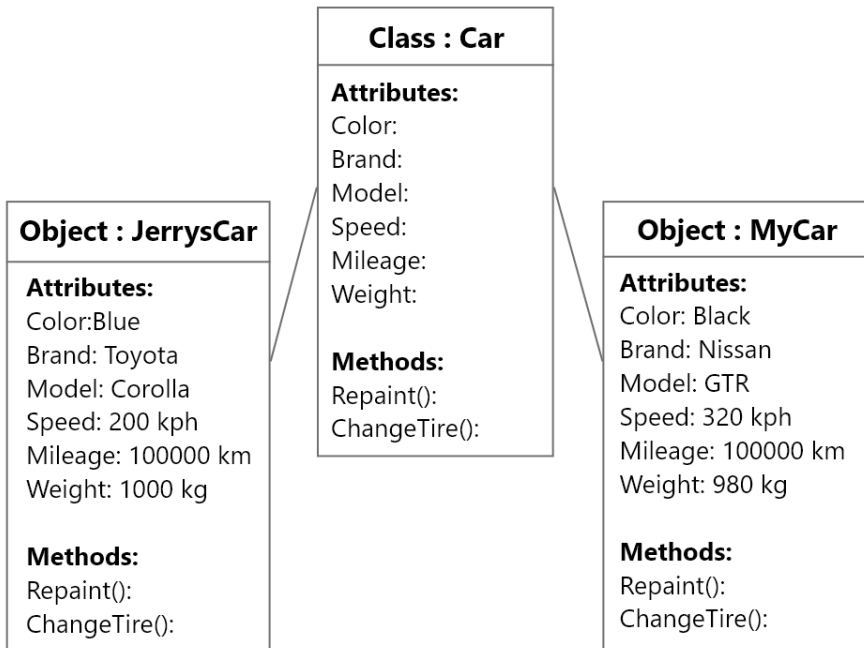
সহজ কথায় জাভা একটি প্রোগ্রামিং ল্যাঙ্গুয়েজ বা ভাষা। এর বিশেষত্ব হল এটি একটি অবজেক্ট ওরিয়েন্টেড প্রোগ্রামিং ল্যাঙ্গুয়েজ। শুধু তাই নয় এটি সম্পূর্ণ অবজেক্ট ওরিয়েন্টেড প্রোগ্রামিং ল্যাঙ্গুয়েজ। এখন স্বাভাবিকভাবেই প্রশ্ন জন্মাবে এই অবজেক্ট ওরিয়েন্টেড প্রোগ্রামিং আবার কি? অবজেক্ট মানে বস্তু সেটা আমরা সকলেই জানি। অবজেক্ট ওরিয়েন্টেড প্রোগ্রামিং সম্পর্কে পরের অধ্যায়ে বিস্তারিত বলব। বস্তু ভিত্তিক যে প্রোগ্রামিং তাকেই বলা হয় অবজেক্ট ওরিয়েন্টেড প্রোগ্রামিং। আমরা আশেপাশে যা দেখি সবই বস্তু। প্রোগ্রামিং করতে গেলে আমাদের আশেপাশের এসব বস্তু নিয়েও কিন্তু প্রোগ্রামিং করতে হয়। যেমন একটি টিকিট বিক্রির সফটওয়্যার। এখানে টিকিট একটি বস্তু, বাস বা ট্রেন বা বিমান একটি বস্তু, যাত্রী একটি বস্তু। এরকম সবকিছুই কোন না কোন বস্তু। অবজেক্ট ওরিয়েন্টেড প্রোগ্রামিং এর মাধ্যমে আমরা এই ধরনের বস্তুগুলোর জন্য অনেক সুন্দর সাজানো এবং বোধগম্য প্রোগ্রাম লিখতে পারি। জাভা যেহেতু সম্পূর্ণ অবজেক্ট ওরিয়েন্টেড প্রোগ্রামিং ল্যাঙ্গুয়েজ তাই আমরা জাভা শিখতেই পারি এবং এটা দিয়ে সুন্দর, সাজানো ও বোধগম্য প্রোগ্রামিং করতে পারি। আর জাভা একটি হাই লেভেল ল্যাঙ্গুয়েজ তাই অনেক কিছুর কোড আগে থেকেই আমাদের জন্য করে দেওয়া আছে। সেজন্য জাভা দিয়ে প্রোগ্রামিং করা অনেকটা সহজ হবে আমাদের জন্য। আরও অনেক ল্যাঙ্গুয়েজ আছে যারা অবজেক্ট ওরিয়েন্টেড প্রোগ্রামিং সাপোর্ট করে যেমন c++, python, c# ইত্যাদি, এসবও আমরা শিখতে পারি। এই ল্যাঙ্গুয়েজ গুলো ও বেশ জনপ্রিয়। তবে আমার ব্যক্তিগত ভাবে অবজেক্ট ওরিয়েন্টেড প্রোগ্রামিং ল্যাঙ্গুয়েজ হিসেবে জাভা কে অনেক গোছানো ভাষা মনে হয়। জাভা আমার অন্যতম পছন্দের একটি ল্যাঙ্গুয়েজ। জাভা প্রোগ্রামের আরেকটি বিশেষ সুবিধা হল এটি যেকোনো অপারেটিং সিস্টেমে বিন্দু বা লেখা হোক না কেন এটি যেকোনো অপারেটিং সিস্টেমে রান হতে সক্ষম। কারন জাভা প্রোগ্রাম রান হয় একটি ভার্চুয়াল মেশিনে (Java Virtual Machine- JVM)। তাই জাভা প্রোগ্রামটি যেই প্ল্যাটফর্ম এই বিন্দু করা হোক বা লেখা হোক না কেন, প্রোগ্রামটি যেকোনো অপারেটিং সিস্টেম এ চলবে যদি

সেই ডিভাইস এ Java Runtime Environment সেট আপ করা থাকে। তাছাড়া জাভা ওয়েব, মোবাইল, ডেস্কটপ অ্যাপ্লিকেশন তৈরির জন্য বেশ সুবিধাজনক।

# অবজেক্ট ওরিয়েন্টেড প্রোগ্রামিং

অবজেক্ট ওরিয়েন্টেড প্রোগ্রামিং হল এমন একটি প্রোগ্রামিং মডেল যার মূল ভিত্তি হল ক্লাস এবং অবজেক্ট। সফটওয়্যার ডিজাইন করার সময় আমরা ক্লাস এবং অবজেক্ট এর উপর অধিক গুরুত্ব দেই। এর ফলে ডেভেলপমেন্ট সহজ হয়ে যায়, সুন্দরভাবে কোডগুলো সাজানো যায়, সবার কাছে কোডগুলো বোধগম্য হয় এবং পরবর্তীতে রক্ষণাবেক্ষণ বা সফটওয়্যার এ কোন পরিবর্তন আনতে সুবিধা হয়।

ক্লাস হচ্ছে একটি ছাঁচ বা blue print যার এক বা একাধিক বৈশিষ্ট্য বা attribute , মেথড বা ফাংশন থাকে এবং এই ছাঁচ ব্যবহার করে একাধিক অবজেক্ট তৈরি করা যায়। আর অবজেক্ট গুলো তৈরি হয় এই ছাঁচ ব্যবহার করে যেখানে attribute গুলোর নির্দিষ্ট মান দেওয়া থাকে। নিচে একটি উদাহরণের মাধ্যমে বোঝানোর চেষ্টা করি।

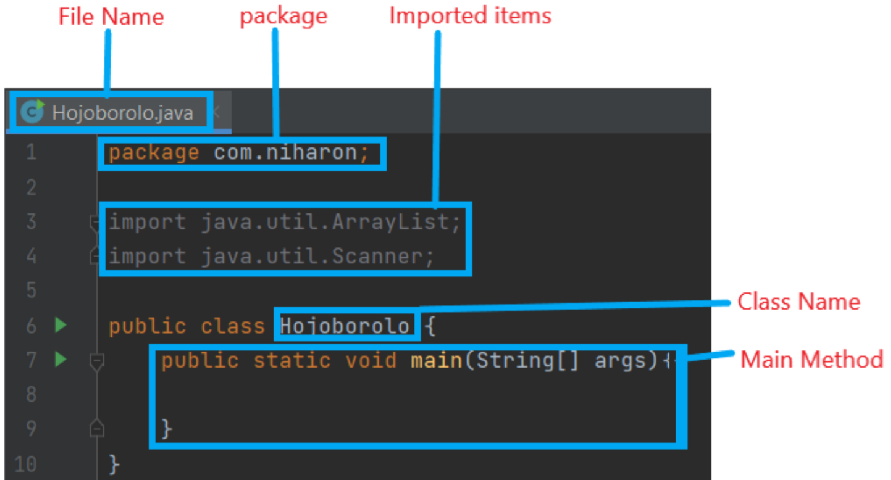


অবজেক্ট ওরিয়েন্টেড প্রোগ্রামিং এর চারটি মূলনীতি আছে। এগুলো হল -

- ১। ইনহেরিটেন্স (Inheritance)
- ২। এনকেপ্সুলেশন (Encapsulation)
- ৩। পলিমরফিজম (Polymorphism)
- ৪। অ্যাবস্ট্রাকশন (Abstraction)

জাভায় এই চারটি মূলনীতির সবগুলোই রয়েছে। তাই জাভাকে সম্পূর্ণ অবজেক্ট ওরিয়েন্টেড প্রোগ্রামিং ল্যঙ্গুয়েজ বলা হয়। এই বইএ ইনহেরিটেন্স, এনকেপ্সুলেশন এবং পলিমরফিজম নিয়ে বিস্তারিত আলোচনা করা হয়েছে। অ্যাবস্ট্রাকশন এই বইএর দ্বিতীয় খণ্ডে পাওয়া যাবে।

# জাভা ফাইলের গঠন



এটি হচ্ছে একদম বেসিক জাভা ফাইলের গঠন। এখানে আমরা দেখতে পারছি যে ফাইল এর নাম এবং ক্লাসের নাম একই। ক্লাসের ভিতরে একটি main মেথড থাকে। ফাংশন কে জাভায় মেথড বলা হয়। শুরুর দিকে আমরা এই main মেথড এর ভিতরেই কোড লিখব। main মেথড এর বাইরেও লিখা যায়। main মেথড এর বাইরে আরও মেথড তৈরি করা যায়। আরও variable ডিক্লেয়ার করা যায়। এগুলো আমরা ধীরে ধীরে শিখব। একেবারে উপরে package name। প্যাকেজ আসলে একটি ফোল্ডার। ক্লাসগুলোকে বিভিন্ন প্যাকেজের মধ্যে রাখা যায়। কোন কিছু একটি জাভা প্রোগ্রামে ব্যবহার করতে হলে তাকে import করতে হয়।

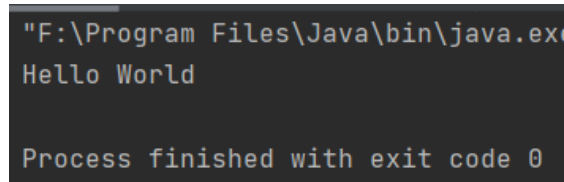
# প্রিন্ট করা

আমরা Hello World প্রিন্ট করা শিখব প্রথমে। প্রায় সব প্রোগ্রামারের প্রোগ্রামিং শুরু হয় Hello World লেখার প্রোগ্রাম এর মাধ্যমে। আমরাও সেই ঐতিহ্য বজায় রাখি।

## Main.java

```
public class Main {  
    public static void main(String[] args) {  
        System.out.println("Hello World");  
    }  
}
```

কোডটি রান করলে নিচের output পাবো।



```
"F:\Program Files\Java\bin\java.exe"  
Hello World  
  
Process finished with exit code 0
```

Println ব্যবহার করার জন্য একটি নতুন লাইন প্রিন্ট হবে। শুধু print ব্যবহার করলে এমনটা হতো না।

## Main.java

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

```
    System.out.print("Hello World");  
}  
}
```

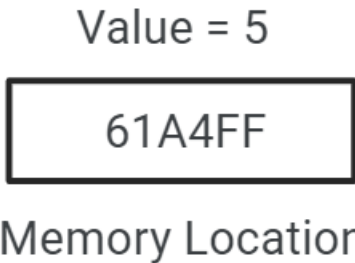
কোডটি রান করলে নিচের output পাবো।

```
"F:\Program Files\Java\bin\java.exe  
Hello World|  
Process finished with exit code 0
```

# ডাটা টাইপস

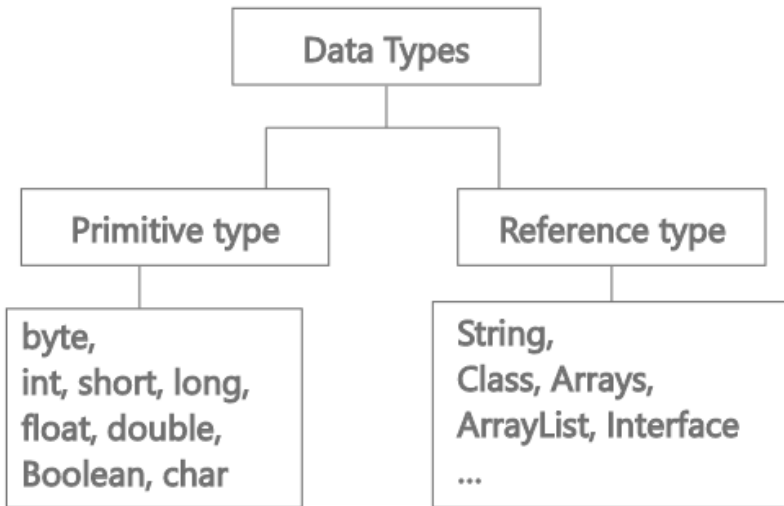
প্রোগ্রাম করতে গেলে আমাদের সংখ্যা, অক্ষর, শব্দ, বাক্য ইত্যাদি নিয়ে কাজ করতে হয়। সংখ্যাও আবার অনেক ধরনের হয়। কোন ধরনের ডাটা নিয়ে কাজ করব তা আমাদের কম্পাইলার কে বলে দিতে হয়। আর এটা বলে দেওয়ার উপায় টাই হল ডাটা টাইপ। জাভায় মূলত দুই ধরনের ডাটা টাইপ আছে। একটি হচ্ছে প্রিমিটিভ টাইপ, অন্যটি নন প্রিমিটিভ বা রেফারেন্স টাইপ। আমরা যারা C, C++ ল্যাপ্সুয়েজ এর সাথে পরিচিত, তারা যে ডাটা টাইপ গুলো ব্যবহার করে এসেছি (int, float, double, long long int, unsigned long long) এসব ই ছিল প্রিমিটিভ টাইপ।

প্রিমিটিভ টাইপ নিয়ে কিছু বলি,  
যদি আমরা একটি integer ডিক্লেয়ার করি এভাবে `int x = 5`; তাহলে মেমরি তে এর representation হবে অনেকটা এরকম,



উপরের চিত্রে লক্ষ্য করলে দেখতে পারব 61A4FF এই মেমরি লোকেশন এ x এর মান রাখা আছে।





এখন আমরা প্রিমিটিভ ডাটা টাইপ গুলোর declaration দেখব। রেফারেন্স টাইপ বা নন-প্রিমিটিভ টাইপ নিয়ে পরবর্তীতে আলোচনা করা হবে।

## Primitive types:

Primitive type variable ডিক্লেয়ার করার সাধারণ নিয়ম:

Type variable = value;

byte:

byte s = 127;

এখানে s হচ্ছে একটি byte টাইপ variable যার ভ্যালু হচ্ছে 127।

byte 1 byte জায়গা দখল করে।

-128 থেকে 127 পর্যন্ত সংখ্যা রাখা যায়।

short:

short x = 10000;

এভাবে short ডিক্লেয়ার করতে হয়। short 2 byte জায়গা দখল করে।

-32,768 থেকে 32,767 পর্যন্ত সংখ্যা রাখা যায়।

int:

int z = 10000000000;

এভাবে int ডিক্লেয়ার করতে হয়। int 4 byte জায়গা দখল করে।

-2,147,483,648 থেকে 2,147,483,647 পর্যন্ত সংখ্যা রাখা যায়।

long:

long y = 10000000000000000000L;

এভাবে long টাইপ ভেরিয়েবল ডিক্লেয়ার করতে হয়। সংখ্যাটির শেষে capital L দিতে হবে অন্যথায় জাভা কম্পাইলার এটিকে বুঝতে পারবে না। কম্পাইলার ধরে নিবে এটি একটি int.

long 8 byte জায়গা দখল করে।

-9,223,372,036,854,775,808 থেকে 9,223,372,036,854,775,807 পর্যন্ত সংখ্যা রাখা যায়।

**double:**

`double x = 5.34;`

ভগ্নাংশ গুলো কে double এ রাখতে হয়। double 8 byte জায়গা দখল করে।  
15 decimal digits পর্যন্ত সংখ্যা রাখা যায়।

**float:**

`float x = 5.34f;`

ভগ্নাংশ গুলো কে float এও রাখা যায়। float 4 byte জায়গা দখল করে। float এর বেলায় সংখ্যাটির শেষে small f অথবা capital F দিতে হবে অন্যথায় জাভা কম্পাইলার এটিকে বুঝতে পারবে না। কম্পাইলার ধরে নিবে এটি একটি double. কারণ ভগ্নাংশ পেলেই জাভা কম্পাইলার ধরে নেয় এটি একটি double।

6 থেকে 7 decimal digits পর্যন্ত সংখ্যা রাখা যায়।

**boolean:**

`boolean b = false;`

boolean এ শুধুমাত্র true/false রাখা যায়। এটি মাত্র 1 bit জায়গা দখল করে।

char:

```
char c = 'A';
```

char এ একটি character রাখা যায়। এটি 2 byte জায়গা দখল করে।

## Reference Type:

String:

String যদিও একটি নন-প্রিমিটিভ ডাটা টাইপ, তবুও প্রয়োজন হবে বলে এখন কিছুটা ধারণা দিচ্ছি।

```
String name = "Abduz Zami";
```

এভাবে আমরা একটি টেম্পট রাখতে পারি। পরবর্তীতে String নিয়ে বিস্তারিত আলোচনা করা হবে।

Class:

ক্লাস একটি রেফারেন্স টাইপ ডাটা টাইপ। ক্লাস নিয়ে পরবর্তীতে আলোচনা করা হবে।

Interface:

ইন্টারফেস একটি রেফারেন্স টাইপ ডাটা টাইপ। Interface নিয়ে পরবর্তীতে আলোচনা করা হবে।

উপরের ডাটা টাইপ গুলোর value প্রিন্ট করে কিভাবে সেটা দেখি।

একটি কোড লিখে ফেলি।

### **Main.java**

```
public class Main {  
    public static void main(String[] args) {  
        int i = 10;  
        byte b = 12;  
        short s = 1000;  
        long l = 14565L;  
        double d = 3.45;  
        float f = 3.1416f;  
        boolean boo = false;  
        char c = 'A';  
        String str = "Abduz Zami";  
  
        System.out.println(i);  
        System.out.println(b);  
        System.out.println(s);  
        System.out.println(l);  
        System.out.println(d);  
        System.out.println(f);  
        System.out.println(boo);  
        System.out.println(c);  
        System.out.println(str);  
    }  
}
```

```

}
}

```

Output টি হবে এরকম

```

"F:\Program Files\Java\bin\java.exe
10
12
1000
14565
3.45
3.1416
false
A
Abduz Zami

Process finished with exit code 0

```

আমরা এটি সুন্দরভাবে প্রিন্ট করব এবার।

কোড এ কিছু পরিবর্তন করে ফেলি।

## Main.java

```

public class Main {
    public static void main(String[] args) {
        int i = 10;
        byte b = 12;
        short s = 1000;
        long l = 14565L;
        double d = 3.45;
    }
}

```

```
float f = 3.1416f;  
boolean boo = false;  
char c = 'A';  
String str = "Abduz Zami";
```

```
System.out.println("Value of int variable: "+i);  
System.out.println("Value of byte variable: "+b);  
System.out.println("Value of short variable: "+s);  
System.out.println("Value of long variable: "+l);  
System.out.println("Value of double variable: "+d);  
System.out.println("Value of float variable: "+f);  
System.out.println("Value of boolean variable: "+boo);  
System.out.println("Value of char variable: "+c);  
System.out.println("Value of String variable: "+str);
```

```
}
```

```
}
```

এর output এমন হবে

```
"F:\Program Files\Java\bin\java.exe" "  
Value of int variable: 10  
Value of byte variable: 12  
Value of short variable: 1000  
Value of long variable: 14565  
Value of double variable: 3.45  
Value of float variable: 3.1416  
Value of boolean variable: false  
Value of char variable: A  
Value of String variable: Abduz Zami  
  
Process finished with exit code 0
```

এখানে আমরা value গুলোর সাথে একটি String যুক্ত করে দিয়েছি।

System.out.println() মেথডটি একটি String গ্রহন করে। আমরা + এর সাহায্যে একাধিক String কে যুক্ত করে একটি String বানাতে পারি। এক্ষেত্রে অন্য টাইপের ডাটা কেউ implicitly String এ type casting করে ফেলেছে। যেমন i কিন্তু int টাইপের variable। এখানে int কে String এ casting করে ফেলেছে। এমনকি শুধু i প্রিন্ট করলেও implicit type casting হয়ে যাচ্ছে।



# ইউজার থেকে ইনপুট নেওয়া

ইউজার ইনপুট নেওয়ার জন্য আমাদের Scanner ক্লাসের একটি অবজেক্ট declare করতে হবে।

## Main.java

```
import java.util.Scanner;
```

```
public class Main{
```

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

```
        Scanner scanner = new Scanner(System.in);
```

```
    }
```

```
}
```

এখানে import java.util.Scanner এর মাধ্যমে java প্যাকেজ এর util প্যাকেজ এর Scanner ক্লাস কে import করা হয়েছে। এভাবেই java তে কোন ক্লাসকে import করতে হয়। একই প্যাকেজ হলে import এর প্রয়োজন নেই। ভিন্ন প্যাকেজ হলে import করতে হবে।

আমরা এখন Scanner এর অবজেক্ট scanner দিয়ে টার্মিনাল থেকে ইনপুট নিব।

আলাদা আলাদা টাইপের ডাটা ইনপুট নেওয়ার জন্য আলাদা আলাদা মেথড ব্যবহার করতে হবে।

আমরা int দিয়ে শুরু করি।

### **Main.java**

```
import java.util.Scanner;
```

```
public class Main{
```

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

```
        Scanner scanner = new Scanner(System.in);
```

```
        int x ;
```

```
        x = scanner.nextInt();
```

```
    }
```

```
}
```

এক লাইনেও করতে পারতাম।

### **Main.java**

```
import java.util.Scanner;
```

```
public class Main{
```

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

```
        Scanner scanner = new Scanner(System.in);
```

```
        int x = scanner.nextInt();
```

```
    }
```

```
}
```

এই x এর মান প্রিন্ট করে দেখতে পারি আমরা।

## Main.java

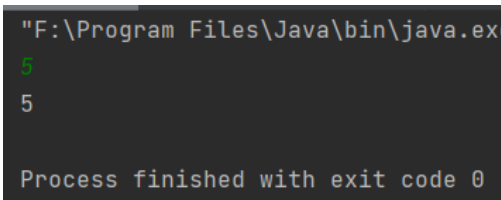
```
import java.util.Scanner;
```

```
public class Main{
```

```
    public static void main (String[] args) {
        Scanner scanner = new Scanner(System.in);
        int x = scanner.nextInt();
        System.out.println(x);
    }
```

```
}
```

কোডটি রান করলে নিচের আউটপুট পাবো।



```
"F:\Program Files\Java\bin\java.exe
5
5
Process finished with exit code 0
```

এখন বাকি গুলোর দেখি।

## Main.java

```
import java.util.Scanner;

public class Main{

    public static void main (String[] args) {
        Scanner scanner = new Scanner(System.in);

        int i = scanner.nextInt();
        float f = scanner.nextFloat();
        double d = scanner.nextDouble();
        byte b = scanner.nextByte();
        boolean boo = scanner.nextBoolean();

        System.out.println(i);
        System.out.println(f);
        System.out.println(f);
        System.out.println(d);
        System.out.println(b);
        System.out.println(boo);
    }
}
```

উপরের সব গুলোই প্রায় এক রকম।

এখন char এর ইনপুট নেওয়া দেখি। char এর ইনপুট নেওয়াটা কিছুটা জটিল। কারণ সরাসরি কোনো মেথড নেই। next() দিয়ে করতে হয়, যা আসলে String ইনপুট নেওয়ার মেথড।

কোড দেখি আমরা।

### **Main.java**

```
import java.util.Scanner;

public class Main{

    public static void main (String[] args) {
        Scanner scanner = new Scanner(System.in);

        char c = scanner.next().charAt(0);
        System.out.println(c);
    }
}
```

ইউজার প্রদত্ত String এর প্রথম character টি c তে যাবে।  
nextLine() দিয়েও করা যায়।

### **Main.java**

```
import java.util.Scanner;
```

```
public class Main{

    public static void main (String[] args) {
        Scanner scanner = new Scanner(System.in);

        char c = scanner.nextLine().charAt(0);
        System.out.println(c);
    }
}
```

এবার দেখব String ইনপুট নেওয়া।

String ইনপুট নেওয়ার দুইটি মেথড আছে - `nextLine()` এবং `next()` ।

শুরুতে `nextLine()` এর ব্যবহার দেখি। `nextLine()` new line অথবা line break অথবা enter এর আগ পর্যন্ত গ্রহন করে। এর সাহায্যে space ও ইনপুট নেওয়া যায়।

আমরা কোড করে দেখি।

### **Main.java**

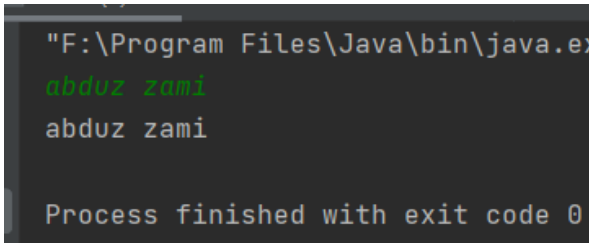
```
import java.util.Scanner;

public class Main{

    public static void main (String[] args) {
        Scanner scanner = new Scanner(System.in);
```

```
String str = scanner.nextLine();
System.out.println(str);
}
}
```

এর output:



```
"F:\Program Files\Java\bin\java.exe"
abduz zami
abduz zami

Process finished with exit code 0
```

এবার next() দিয়ে দেখি। next() space, new line, line break, enter এর আগ পর্যন্ত গ্রহণ করে। এর সাহায্যে space ইনপুট নেওয়া যায়না। কারন এটি space পেলে break করে।

আমরা কোড করে ফেলি।

## Main.java

```
import java.util.Scanner;
```

```
public class Main{
```

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

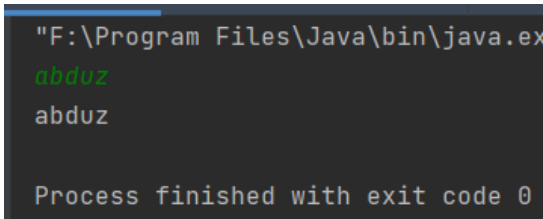
```
        Scanner scanner = new Scanner(System.in);
```

```

String str = scanner.next();
System.out.println(str);
}
}

```

এর output:



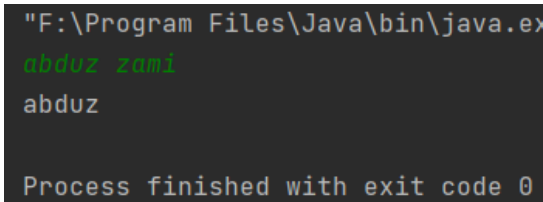
```

"F:\Program Files\Java\bin\java.exe"
abduz
abduz

Process finished with exit code 0

```

এখানে যদি `abduz zami` ইনপুট দিতাম তাহলে শুধু `abduz` ই গ্রহণ করত।



```

"F:\Program Files\Java\bin\java.exe"
abduz zami
abduz

Process finished with exit code 0

```

কারণ `next()` space পেলে ব্রেক হয়।

## Scanner.nextLine() ব্যবহারের সতর্কতা

### Main.java

```
import java.util.Scanner;
```



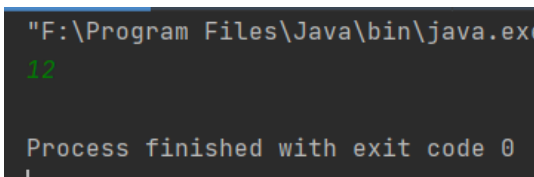
```

public class Main{

    public static void main (String[] args) {
        Scanner scanner = new Scanner(System.in);

        int i = scanner.nextInt();
        String s = scanner.nextLine();
    }
}

```



```

"F:\Program Files\Java\bin\java.exe
12
Process finished with exit code 0

```

এভাবে যদি int ইনপুট নেওয়ার পর `nextLine()` দিয়ে String ইনপুট নিতে যাই তাহলে String এ null value assign হবে। কারণ `nextInt()` এ আমরা enter প্রেস করেছিলাম। `nextInt()` int ছাড়া অন্য কিছু পেলে ব্রেক করে। তাই new line বা enter টি stream এ থেকে যায়। যা পরের `nextLine()` গ্রহণ করে। এই new line এর আগে যেহেতু কোন কিছু নেই সেহেতু String টি তে null value assign হয়। এখন আবার যদি `nextLine` এর পরে `nextInt` বা অন্য কোন মেথড নেই `nextLine` বাদে তাহলে error পেতে পারি।

যেমন যদি নিচের কোডটি দেখি।

### Main.java

```

import java.util.Scanner;

```

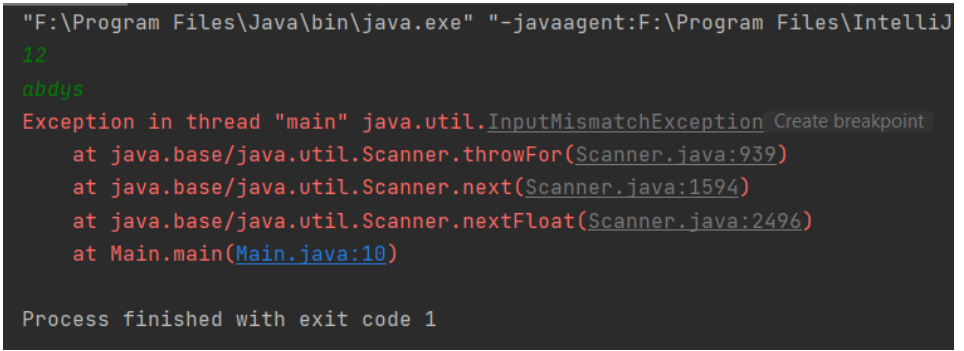
```

public class Main{

    public static void main (String[] args) {
        Scanner scanner = new Scanner(System.in);

        int i = scanner.nextInt();
        String s = scanner.nextLine();
        float f = scanner.nextFloat();
    }
}

```



```

"F:\Program Files\Java\bin\java.exe" "-javaagent:F:\Program Files\IntelliJ
12
abdys
Exception in thread "main" java.util.InputMismatchException Create breakpoint
    at java.base/java.util.Scanner.throwFor(Scanner.java:939)
    at java.base/java.util.Scanner.next(Scanner.java:1594)
    at java.base/java.util.Scanner.nextFloat(Scanner.java:2496)
    at Main.main(Main.java:10)

Process finished with exit code 1

```

এখানে exception দেখানোর কারন হল। 12 ইনপুট দেওয়ার সময় যে অতিরিক্ত new line বা enter stream এ রয়ে গিয়েছিল সেটি nextLine() কর্তৃক গ্রহীত হয়েছে। যার কারনে ইউজার প্রদত্ত String abdys কে nextFloat() দিয়ে ইনপুট নেওয়া যাচ্ছে না। কারন abdys একটি String, float নয়।

এটা শুধু nextInt বা nextFloat এর জন্য প্রযোজ্য নয়। nextLine বাদে অন্য যেকোনো Scanner method এর জন্য প্রযোজ্য।

এর থেকে রক্ষা পাওয়ার উপায় দেখি আমরা।

খুব বেশি কিছু নয়। nextInt() nextFloat() next() এসবের পরে একটি nextLine() দিয়ে দিলেই অতিরিক্ত new line বা enter টি vanish হয়ে যাবে।

## Main.java

```
import java.util.Scanner;

public class Main{

    public static void main (String[] args) {
        Scanner scanner = new Scanner(System.in);

        int i = scanner.nextInt();
        scanner.nextLine();
        String s = scanner.nextLine();
        float f = scanner.nextFloat();
        scanner.nextLine();

        System.out.println(i);
        System.out.println(s);
        System.out.println(f);
    }
}
```

এর output যদি দেখি।

```
"F:\Program Files\Java\bin\java.exe"
12
abduz zami
3.1416
12
abduz zami
3.1416

Process finished with exit code 0
```

আমরা নিচের কাজ টিও করতে পারতাম।

### Main.java

```
import java.util.Scanner;
```

```
public class Main{
```

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

```
        Scanner scanner = new Scanner(System.in);
```

```
        int i = Integer.parseInt(scanner.nextLine());
```

```
        String s = scanner.nextLine();
```

```
        float f = Float.parseFloat(scanner.nextLine());
```

```
System.out.println(i);  
System.out.println(s);  
System.out.println(f);  
}  
}
```

Integer.parseInt এর কাজ হচ্ছে String কে int এ রূপান্তর করা। আর  
Integer.parseFloat এর কাজ হচ্ছে String কে float এ রূপান্তর করা।

# ফরমেট স্পেসিফায়ার

ফরমেট স্পেসিফায়ার হচ্ছে কোন ডাটা টাইপের নির্দেশক।

জাভায় বহুল ব্যবহৃত কয়েকটি ফরমেট স্পেসিফায়ার নিয়ে আলোচনা করছি।

Format Specifier	What it means
%d	Int, byte, short, long
%f	Float, double
%c	char
%C	char (capitalized)
%s	String
%S	String capitalized
%b	boolean (true/false)
%B	boolean capitalized (TRUE/FALSE)
%e	Scientific notation (e)

%E	Scientific notation (E)
%g	Either decimal or scientific (e) is small
%G	Either decimal or scientific (E) is capital
%h	HashCode of argument not memory address
%H	HashCode (capitalized)
%x	Hexadecimal Value
%X	Hexadecimal Value (capitalized)
%a	Floating point hexadecimal
%A	Floating point hexadecimal (capitalized)

এদের ব্যবহার দেখি আমরা।

এদের কে printf এর সাহায্যে ব্যবহার করা যায়। আবার Formatter এর সাহায্যেও ব্যবহার করা যায়। আমরা দুটোই দেখব। শুরুতে printf এর সাহায্যে দেখি।

**%d**

**Main.java**

```
public class Main {
```

```
public static void main(String[] args) {
    System.out.printf("%d", 1000000);
}
}
```

Output:

1000000

এখানে “ “ এর ভিতরের %d , এর পরের সংখ্যাটিকে নির্দেশ করছে। আমরা সরাসরি সংখ্যা না দিয়ে variable ও দিতে পারি।

### **Main.java**

```
public class Main {
    public static void main(String[] args) {
        int x = 1000000;
        System.out.printf("%d", x);
    }
}
```

Output:

1000000

Byte, short, long এর জন্য একই ফরম্যাট স্পেসিফায়ার ব্যবহার করা হয়।

### **Main.java**

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



```
byte x = 125;
System.out.printf("%d",x);
}
}
```

Output:

125

### **Main.java**

```
public class Main {
    public static void main(String[] args) {
        short x = 12564;
        System.out.printf("%d",x);
    }
}
```

Output:

12564

### **Main.java**

```
public class Main {
    public static void main(String[] args) {
        System.out.printf("%d",1256245445224542441L);
    }
}
```

Output:

1256245445224542441

### **Main.java**

```
public class Main {
    public static void main(String[] args) {
        long x = 1256245445224542441L;
        System.out.printf("%d",x);
    }
}
```

Output:

1256245445224542441

**%f**

float আর double এর ফরম্যাট স্পেসিফায়ার একই %f।

### **Main.java**

```
public class Main {
    public static void main(String[] args) {
        float x = 12.35f;
        System.out.printf("%f",x);
    }
}
```

Output:

12.350000

আমরা চাইলে দশমিকের পর কত ঘর প্রিন্ট হবে ঠিক করে দিতে পারি। এর জন্য `f` এর আগে `.` দিয়ে কত ঘর প্রিন্ট করব বলে দিতে হবে।

### Main.java

```
public class Main {
    public static void main(String[] args) {
        float x = 12.35f;
        System.out.printf("%.2f",x);
    }
}
```

Output:

12.35

### Main.java

```
public class Main {
    public static void main(String[] args) {
        double x = 12.35;
        System.out.printf("%f",x);
    }
}
```

Output:

12.350000

**Main.java**

```
public class Main {
    public static void main(String[] args) {
        double x = 12.35;
        System.out.printf("%.2f",x);
    }
}
```

Output:

12.35

**%C**

char এর ফরম্যাট স্পেসিফায়ার হচ্ছে %c।

**Main.java**

```
public class Main {
    public static void main(String[] args) {
        char x = 'a';
        System.out.printf("%c",x);
    }
}
```

Output:

a

Capital C ব্যবহার করলে char variable টি capitalized হয়ে যাবে।

**%C**

**Main.java**

```
public class Main {
    public static void main(String[] args) {
        char x = 'a';
        System.out.printf("%C",x);
    }
}
```

Output:

A

**%S**

String এর ফরম্যাট স্পেসিফায়ার হচ্ছে %s

**Main.java**

```
public class Main {
    public static void main(String[] args) {
        String x = "Abduz Zami";
        System.out.printf("%S",x);
    }
}
```

```
}
```

Output:

Abduz Zam

## %S

Capital %S ব্যবহার করলে String টি capitalized হয়ে যাবে।

**Main.java**

```
public class Main {
    public static void main(String[] args) {
        String x = "Abduz Zami";
        System.out.printf("%S",x);
    }
}
```

Output:

ABDUZ ZAMI

## %b

Boolean এর format specifier %b

**Main.java**

```
public class Main {
    public static void main(String[] args) {
        boolean x = false;
        System.out.printf("%b",x);
    }
}
```

Output:

false

### **Main.java**

```
public class Main {
    public static void main(String[] args) {
        boolean x = true;
        System.out.printf("%b",x);
    }
}
```

Output:

true

আমরা চাইলে , এর পরে কোন condition যাচাই করে সত্য না মিথ্যা দেখতে পারি।

### **Main.java**

```
public class Main {
    public static void main(String[] args) {
        System.out.printf("%b",5>6);
    }
}
```

```
}
```

Output:

false

### **Main.java**

```
public class Main {
    public static void main(String[] args) {
        System.out.printf("%b",5<6);
    }
}
```

Output:

true

## **%B**

%B ব্যবহার করলে TRUE / FALSE capitalized হয়ে প্রিন্ট হবে।

### **Main.java**

```
public class Main {
    public static void main(String[] args) {
        System.out.printf("%B",5<6);
    }
}
```



Output:

TRUE

**%e**

%e দিয়ে সাইন্টিফিক ভাবে প্রিন্ট করা যায়।

**Main.java**

```
public class Main {  
    public static void main(String[] args) {  
        System.out.printf("%e",102.26);  
    }  
}
```

Output:

1.022600e+02

**%E**

%E ব্যবহার করলে e টা E হয়ে যাবে,

**Main.java**

```
public class Main {  
    public static void main(String[] args) {  
        System.out.printf("%E",102.26);  
    }  
}
```

```
}
}
```

Output:

1.022600E+02

**%g**

%g দশমিক বাঁ সাইন্টিফিক যেকোনো একটি পদ্ধতিতে প্রিন্ট করে। সংখ্যাটি যদি  $10^6$  এর থেকে ছোট হয়ে তাহলে দশমিক পদ্ধতিতে প্রিন্ট করে। অন্যথায় সাইন্টিফিক পদ্ধতিতে।

**Main.java**

```
public class Main {
    public static void main(String[] args) {
        System.out.printf("%g", 10256.2657);
    }
}
```

Output:

10256.3

**Main.java**

```
public class Main {
    public static void main(String[] args) {
        System.out.printf("%g", 1025646.2657);
    }
}
```

```
}
```

Output:

1.02565e+06

## %G

%G শুধু E কে capitalized করে।

### Main.java

```
public class Main {
    public static void main(String[] args) {
        System.out.printf("%G", 1025646.2657);
    }
}
```

Output:

1.02565E+06

## %h

%h শুধু মাত্র hashcode প্রিন্ট করে। এটি কোন address নয়। বার বার রান করলেও দেখা যাবে একই value আসছে।

### Main.java

```
public class Main {
```

```

public static void main(String[] args) {
    System.out.printf("%h", 1025646.2657);
}
}

```

Output:

c9269849

### **Main.java**

```

public class Main {
    public static void main(String[] args) {
        System.out.printf("%h", 1025646);
    }
}

```

Output:

fa66e

## **%H**

%H শুধু capitalize করে দিবে character গুলো।

### **Main.java**

```

public class Main {
    public static void main(String[] args) {
        System.out.printf("%H", 1025646);
    }
}

```

```
}
}
```

Output:

FA66E

**%X**

%x দিয়ে ইন্টিজারের hexadecimal value প্রিন্ট করে।

**Main.java**

```
public class Main{

    public static void main (String[] args) {
        System.out.printf("%x",154165201);
    }
}
```

Output:

9305fd1

**%X**

%X দিলে character গুলো capitalized হয়ে যাবে।

**Main.java**

```
public class Main{

    public static void main (String[] args) {
        System.out.printf("%X",154165201);
    }
}
```

Output:

9305FD1

**%a**

%a floating point hexadecimal প্রিন্ট করে। 0x দিয়ে এটা যে একটি হেক্সা ডেসিমাল সংখ্যা টা বোঝায়।

**Main.java**

```
public class Main {
    public static void main(String[] args) {
        System.out.printf("%a",1025.1274445);
    }
}
```

Output:

0x1.0048280cf9e38p10

## %A

%A শুধু character গুলো capitalize হবে।

### Main.java

```
public class Main {
    public static void main(String[] args) {
        System.out.printf("%A", 1025.1274445);
    }
}
```

Output:

0X1.0048280CF9E38P10

## একাধিক ফরম্যাট স্পেসিফায়ার একসাথে

এবার আমরা একাধিক ফরম্যাট স্পেসিফায়ার একসাথে ব্যবহার করব।

### Main.java

```
public class Main {
    public static void main(String[] args) {
        int x = 50, y = 40;
        System.out.printf("%d + %d = %d", x, y, x+y);
    }
}
```

Output:

50 + 40 = 90

এখানে প্রথম %d (,) এর পরের প্রথম variable অর্থাৎ x কে নির্দেশ করছে। পরের টা y। তার পরের টা x+y কে। এখানেই আমরা যোগের কাজ ও করতে পারি। ফরম্যাট স্পেসিফায়ার ছাড়া বাকি character গুলো যেভাবে আছে সেভাবেই প্রিন্ট হবে।

## Formatter class

Formatter class এর সাহায্যে কিভাবে ফরম্যাট করা যায় দেখব আমরা।

### Main.java

```
import java.util.Formatter;

public class Main {
    public static void main(String[] args) {
        Formatter formatter = new Formatter();
        formatter.format("Value : %d",125);
        System.out.println(formatter);
    }
}
```

Output:

Value : 125



**Main.java**

```
import java.util.Formatter;

public class Main {
    public static void main(String[] args) {
        Formatter formatter = new Formatter();
        formatter.format("Sum of %d & %d is : %d", 10, 20, 10+20);
        System.out.println(formatter);
    }
}
```

Output:

Sum of 10 & 20 is : 30

## স্পেসিং ঠিক করা

অনেক সময় আউটপুট সাজানোর জন্য এবং সুন্দর করার জন্য আমাদের padding এবং spacing ঠিক করতে হয়। নিচে স্পেসিং ঠিক করার কয়েকটি উদাহরণ দেওয়া হল।  
প্রথমে int এর জন্য দেখি।

**Main.java**

```
public class Main{
```

```

public static void main (String[] args) {
    int y = 123;
    System.out.printf("%5d",y);
}
}

```

এখানে যে কাজ টি হয়েছে সেটি চিত্রের সাহায্যে বোঝানোর চেষ্টা করি। এখানে 123 এই int টি 5 ঘর জায়গা নিবে। অর্থাৎ 123 তিন ঘর জায়গা নেওয়ার পরেও আরও দুইটি স্পেস অতিরিক্ত নিবে।



### Main.java

```

public class Main{

    public static void main (String[] args) {
        int y = 123, z = 10;
        System.out.printf("%5d %6d",y,z);
    }
}

```

এখানে 123 পাঁচ ঘর এবং 10 ছয় ঘর জায়গা নিয়েছে।

		1	2	3					1	0
--	--	---	---	---	--	--	--	--	---	---

double এবং float এর বেলায় দশমিক এর পর ঘর সংখ্যা ঠিক করে না দিলে default ছয় ঘর নিবে দশমিক এর পর। নিচের উদাহরণ টিতে y এর জন্য দশমিকের পর ছয় ঘর 345000 এবং 12 মত আট ঘর নিয়েছে এবং দুইটি স্পেস নিয়েছে। z এর বেলায় দশমিকের পর ঘর সংখ্যা ঠিক করে দেওয়া হয়েছে। নিচের কোডটি দেখি।

### Main.java

```
public class Main{

    public static void main (String[] args) {
        double y = 12.345, z = 10.235;
        System.out.printf("%10f %8.3f",y,z);
    }
}
```

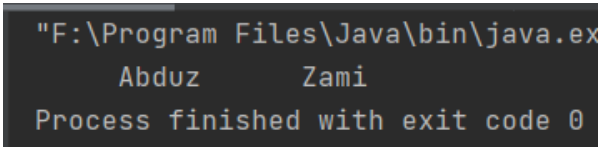
Output:

```
"F:\Program Files\Java\bin\java.exe
12.345000    10.235
Process finished with exit code 0"
```

এখন স্ট্রিং এর জন্য দেখি।

```
public class Main{  
  
    public static void main (String[] args) {  
        String y = "Abduz", z = "Zami";  
        System.out.printf("%10s",y);  
        System.out.printf("%10s",z);  
    }  
}
```

Output:



```
"F:\Program Files\Java\bin\java.exe"  
    Abduz      Zami  
Process finished with exit code 0
```

# অপারেটর

জাভার অপারেটর গুলোকে নিম্নোক্ত পাঁচ ভাগে ভাগ করা যায়।

- অ্যারিথমেটিক অপারেটর (Arithmetic operator)
- অ্যাসাইনমেন্ট অপারেটর (Assignment operators)
- কম্পারিজন অপারেটর (Comparison operators)
- লজিক্যাল অপারেটর (Logical operators)
- বিটওয়াইজ অপারেটর (Bitwise operators)

## অ্যারিথমেটিক অপারেটর:

Operator	Name	Description	Example
+	Addition	দুইটি সংখ্যা যোগ করে	$x + y$
-	Subtraction	একটি সংখ্যা থেকে অন্য সংখ্যা বিয়োগ করে	$x - y$
*	Multiplication	দুইটি সংখ্যা গুন কর	$x * y$

/	Division	একটি সংখ্যা দ্বারা অন্য একটি সংখ্যা ভাগ করে	$x/y$
%	Modulus	ভাগশেষ প্রদান করে	$x\%y$
++	Increment	কোন চলকের মান এক করে বাড়ায়	$x++$ , $++x$
--	Decrement	কোন চলকের মান এক করে কমায়	$x--$ , $--x$

## অ্যাসাইনমেন্ট অপারেটর:

Operator	Description	Example	Same As
=	= এর ডান পাশের মান বাম পাশের চলকে assign করে	$x=5$	$x=5$
+=	= এর বাম পাশের চলকের মান ডান পাশের মানের	$x+=5$	$x=x+5$

	সমান বাড়ায়		
--=	= এর বাম পাশের চলকের মান ডান পাশের মানের সমান কমায়	$x-=5$	$x=x-5$
*=	= এর বাম পাশের চলকের মান ডান পাশের মানের সাথে গুন করে বাম পাশের চলকে রাখা হয়	$x*=5$	$x=x*5$
/=	= এর বাম পাশের চলকের মান ডান পাশের মান দ্বারা ভাগ করে বাম পাশের চলকে রাখা হয়	$x/=5$	$x=x/5$
%=	= এর বাম পাশের চলকের মান ডান পাশের মান দ্বারা ভাগ করলে যে ভাগশেষ হয় তাকে বাম পাশের চলকে	$x\%=5$	$x=x+5$

	রাখা হয়		
$\&=$	= এর বাম পাশের চলকের সাথে ডান পাশের মান এর AND অপারেশন করে বাম পাশের চলকে রাখা হয়	$x\&=5$	$x=x\&5$
$ =$	= এর বাম পাশের চলকের সাথে ডান পাশের মান এর OR অপারেশন করে বাম পাশের চলকে রাখা হয়	$x =5$	$x=x 5$
$\wedge=$	= এর বাম পাশের চলকের সাথে ডান পাশের মান এর XOR অপারেশন করে বাম পাশের চলকে রাখা হয়	$x\wedge=5$	$x=x\wedge5$
$>>=$	= এর বাম পাশের চলকের সাথে ডান পাশের মান এর right shift	$x>>=2$	$x=x>>2$



	অপারেশন করে বাম পাশের চলকে রাখা হয়		
<<=	= এর বাম পাশের চলকের সাথে ডান পাশের মান এর left shift অপারেশন করে বাম পাশের চলকে রাখা হয়	$x \ll 2$	$x = x \ll 2$
>>=	= এর বাম পাশের চলকের সাথে ডান পাশের মান এর unsigned right shift অপারেশন করে বাম পাশের চলকে রাখা হয়	$x \gg 2$	$x = x \gg 2$

## কম্পারিজন অপারেটর:

Operator	Name	Description	Example
----------	------	-------------	---------

==	Equal to	উভয় এর মান সমান হলে true রিটার্ন করে	$x == y$
!=	Not equal	উভয় এর মান সমান না হলে true রিটার্ন করে	$x != y$
>	Greater than	বামপক্ষ ডানপক্ষ থেকে বড় হলে true রিটার্ন করে	$x > y$
<	Less than	বামপক্ষ ডানপক্ষ থেকে ছোট হলে true রিটার্ন করে	$x < y$
>=	Greater than or equal to	বামপক্ষ ডানপক্ষ থেকে বড় বাঁ সমান হলে true রিটার্ন করে	$x >= y$
<=	Less than or equal to	বামপক্ষ ডানপক্ষ থেকে ছোট বাঁ সমান হলে true রিটার্ন করে	$x <= y$

## লজিক্যাল অপারেটর:

Operator	Name	Description	Example
&&	Logical and	উভয়টি সত্য হলে true রিটার্ন করে	$x < 5 \ \&\& \ x < 10$
	Logical or	যেকোনো একটি সত্য হলে true রিটার্ন করে	$x < 5 \    \ x < 4$
!	Logical not	রেসাল্ট কে উলটিয়ে দেয়। অর্থাৎ true হলে false এবং false হলে true করে দেয়	$!(x < 5 \ \&\& \ x < 10)$

## বিটওয়াইজ অপারেটর:

Operator	Name	Description	Example
&	Bitwise AND	দুইটি সংখ্যার Bitwise	$5 = 101$ $3 = 011$

		AND অপারেশন করে	$5 \& 3 = 001 = 1$
	Bitwise OR	দুইটি সংখ্যার Bitwise OR অপারেশন করে	$5 = 101$ $3 = 011$ $5   3 = 111 = 7$ (Decimal)
^	Bitwise XOR	দুইটি সংখ্যার Bitwise XOR অপারেশন করে	$5 = 101$ $3 = 011$ $5 \wedge 3 = 110 = 6$
~	Bitwise complement	একটি সংখ্যার complement সংখ্যা বের করে	$5 = 00000101$ $\sim 5 = 11111010$ $11111010 = -6$
<<	Left shift	একটি সংখ্যার বিট গুলোকে বাম দিকে শিফট করে।	$5 = 101$ $5 << 2 = 10100$ $10100 = 20$
>>	Right shift	একটি সংখ্যার বিট গুলোকে ডান দিকে শিফট করে।	$5 = 101$ $5 >> 2 = 001 = 1$  $-5 =$ $11111111111111$ $11111111111111$ $1011$

			$-5 \gg 2 =$ $11111111111111$ $11111111111111$ $1110$ $= -2$  বাইনারি সংখ্যায় বেলায় একেবারের বামের বিট টি সাইন বিট। পসিটিভ সংখ্যার বেলায় সাইন বিট টি 0 থাকে। আর নেগেটিভ সংখ্যায় সাইন বিট টি থাকে 1। নেগেটিভ সংখ্যার রাইট শিফট করলে সাইন বিট শিফট হয় না।
$\gg$	Unsigned Right Shift	একটি সংখ্যার বিট গুলোকে ডান দিকে শিফট করে। সাইন বিট সহ	$-5 =$ $11111111111111$ $11111111111111$ $1011$

			$-5 \gg 2 =$ 001111111111 111111111111 11110 =1073741822  Unsigned right shift করলে সাইন বিট টিও শিফট হয়।
--	--	--	---

# কন্ডিশনাল লজিক

কম্পিউটার এর কিন্তু আমাদের মত চিন্তা করার সামর্থ্য নেই। সে সবসময় শর্ত মেনে কাজ করে। সে কি করবে আর কি করবেনা তা ঠিক করে শর্ত গুলো চেক করে। আর এই শর্ত গুলো চেক করার জন্য ব্যবহার করা হয় if-else। তো আমরা কথা না বাড়িয়ে কোডিং এ চলে যাই।

```
public class booktest {
    public static void main(String[] args) {
        if (3>5){
            System.out.println("RAIN");
        }else{
            System.out.println("SUN");
        }
    }
}
```

If এর ভিতরে ৩ কি ৫ থেকে বড় কিনা চেক করছে। যদি সত্য হয় তবে RAIN প্রিন্ট করবে আর যদি মিথ্যা হয় তবে SUN প্রিন্ট করবে।

যেহেতু শর্ত টি মিথ্যা। সেহেতু SUN প্রিন্ট করবে।

আমরা আরেকটি উদাহরন দেখি।

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

```

int n = 10;
if (n>5){
    System.out.println("RAIN");
}else{
    System.out.println("SUN");
}
}
}

```

এখানে একটি int n নিয়েছি। n এর মান 10 । if এর ভিতরে বলা হয়েছে যদি n এর মান ৫ থেকে বড় হয় তবে RAIN প্রিন্ট করবে, না হলে SUN প্রিন্ট করবে। যেহেতু n এর মান ৫ থেকে বড় তাই RAIN প্রিন্ট করবে।

```

public class booktest {
    public static void main(String[] args) {
        char c = 'A';
        if (c=='A'){
            System.out.println("RAIN");
        }else{
            System.out.println("SUN");
        }
    }
}

```

উপরের কোডটিতে একটি char নেওয়া হয়েছে c। c যদি A হয় তবেই শুধু RAIN প্রিন্ট করবে অন্যথায় SUN। অবশ্যই এটি RAIN প্রিন্ট করবে যেহেতু শর্তটি সত্য হয়েছে।



এবার আমরা চেক করব char টি কি A থেকে বড় এবং F থেকে ছোট কিনা।

```
public class booktest {
    public static void main(String[] args) {
        char c = 'D';
        if (c>'A' && c<'F'){
            System.out.println("RAIN");
        }else{
            System.out.println("SUN");
        }
    }
}
```

ছোট অথবা সমান বা বড় অথবা সমান ও চেক করতে পারতাম।

```
public class booktest {
    public static void main(String[] args) {
        char c = 'D';
        if (c>='A' && c<='F'){
            System.out.println("RAIN");
        }else{
            System.out.println("SUN");
        }
    }
}
```

char টি A অথবা D কিনা সেটাও চেক করতে পারতাম। এর জন্য ব্যবহার করব logical or operator। আমরা পূর্ববর্তী অধ্যায়ে শিখেছি। আমরা জানি যে শর্তগুলোর যেকোনো একটি সত্য হলেই লজিকাল অর সত্য হয় বা রিটার্ন করে।

```
public class booktest {
    public static void main(String[] args) {
        char c = 'D';
        if (c=='A' || c=='D'){
            System.out.println("RAIN");
        }else{
            System.out.println("SUN");
        }
    }
}
```

আমরা কোন একটি শর্তের ফল কে উলতিয়ে দিতে পারি। অর্থাৎ যদি শর্তটি সত্য হয় তবে মিথ্যা এবং যদি মিথ্যা হয় তবে সত্য রিটার্ন করে। এই কাজ টি আমরা করতে পারি লজিকাল নট অপারেটর এর সাহায্যে।

```
public class booktest {
    public static void main(String[] args) {
        char c = 'D';
        if (!(c == 'D')){
            System.out.println("RAIN");
        }else{
            System.out.println("SUN");
        }
    }
}
```

}

}

}

এখানে RAIN প্রিন্ট করার কথা থাকলেও প্রিন্ট হবে SUN। কেন হবে সেটা একটু আগেই বলেছি।

# লুপ

ধরি আমাদের ১ থেকে ১০ পর্যন্ত সংখ্যা গুলোকে প্রিন্ট করতে হবে। আমরা কাজটি এভাবে করতে পারি।

```
public class booktest {
    public static void main(String[] args) {
        System.out.println(1);
        System.out.println(2);
        System.out.println(3);
        System.out.println(4);
        System.out.println(5);
        System.out.println(6);
        System.out.println(7);
        System.out.println(8);
        System.out.println(9);
        System.out.println(10);

    }
}
```

কাজটি কষ্টসাধ্য। সংখ্যার পরিমাণ যদি আর বেশি হত তবে কাজটি আর কঠিন হত। আমাদের একই কাজ বার বার করতে হচ্ছে। এর থেকে পরিত্রানের উপায় হচ্ছে লুপ। আর কথা না বাড়িয়ে দেখি লুপ কি জিনিস কিভাবে কাজ করে।

জাভায় চার ধরনের লুপ আছে।

- ১) for loop
- ২) while loop
- ৩) do-while loop
- ৪) for-each loop

## for loop

উপরের ১ থেকে ১০ পর্যন্ত প্রিন্ট করা যদি for লুপের সাহায্যে করতাম আমরা তাহলে এমন হত।

### booktest.java

```
public class booktest {
    public static void main(String[] args) {
        for (int i = 1; i <= 10; i++) {
            System.out.println(i);
        }
    }
}
```

অনেক সহজ হয়ে গেছে। তাই না?

For loop এ কি হয় সেটা এবার জানব আমরা। শুরুতে যেকোনো একটি চলকের প্রারম্ভিক মান ধরে নিতে হবে। এখানে  $i=0$  ধরেছি। এই চলকের মান প্রথমে চেক হবে। দেখবে মানটি শর্ত মানে কিনা। যদি শর্ত মানে তবেই লুপ ঘুরবে। অন্যথায় থেমে যাবে।

এখানে শর্তটি হচ্ছে  $i \leq 10$ । অর্থাৎ  $i$  এর মান ১০ থেকে ছোট হলে লুপ টি ঘুরবে।

প্রত্যেকবার লুপটি ঘুরলে চলকটির মান ৩ বারতে থাকে। আমরা বলে দিতে পারি চলকটির মান কি হারে বারবে।

ধরি এবারর একটি সমান্তর ধারা প্রিন্ট করতে হবে।

১ ৩ ৫ ৬ ৭ ৯

দেখে বুঝা যাচ্ছে এখানে চলকের মান ২ করে বারতে হবে।

### **booktest.java**

```
public class booktest {
    public static void main(String[] args) {
        for (int i = 1; i <= 10; i+=2) {
            System.out.println(i);
        }
    }
}
```

এবার একটি গুনতর ধারা প্রিন্ট করতে হবে।

১ ২ ৪ ৮ ১৬ ৩২

দেখা যাচ্ছে ধারাতির সাধারণ অনুপাত ২। অর্থাৎ চলকের মান দিগুন হারে বারতে হবে।

### **booktest.java**

```
public class booktest {
    public static void main(String[] args) {
        for (int i = 1; i <= 32; i*=2) {
```

```

        System.out.println(i);
    }
}

```

প্রথম ধারাটি যদি উলটা প্রিন্ট করতে হয় তাহলে কি করতে হবে দেখি এবার।

### **booktest.java**

```

public class booktest {
    public static void main(String[] args) {
        for (int i = 10; i > 0; i--) {
            System.out.println(i);
        }
    }
}

```

এখানে চলকের মান এক করে কমানো হয়েছে।  $i > 0$  এর মানে  $i \geq 1$ । সুতরাং,  $i$  এর মান দশ থেকে এক করে কমতে কমতে যখন শূন্য হয়ে যাবে তখন লুপটি থেকে যাবে। অনেকের প্রশ্ন আসতে পারে চলকের মান শূন্য হলে প্রিন্ট হল না কেন। এর কারণ এই যে যখন  $i$  এর মান শূন্য হল তখন আর লুপের ভিতরে ঢুকতেই পারেনি।

## **while loop**

1 2 3 4 5 6 7 8 9 10 এই ধারাটি while loop এর সাহায্যে প্রিন্ট করি।

**booktest.java**

```

public class booktest {
    public static void main(String[] args) {
        int i = 0;
        while(i<=10)
        {
            System.out.println(i);
            i++;
        }
    }
}

```

while loop এ বন্ধনি () এর ভিতরে শর্ত দিয়ে দিতে হয়।

ধারাটি উলটা করে প্রিন্ট করি এবার।

**booktest.java**

```

public class booktest {
    public static void main(String[] args) {
        int i = 10;
        while(i-->0)
        {
            System.out.println(i+1);
        }
    }
}

```



```
}
}
```

এখানে প্রতিবার লুপ ঘুরার সময়  $i$  এর মান এক করে কমছে আর সেই মানটি শূন্য থেকে বড় কিনা চেক হচ্ছে। এখানে  $i--$  করার সময়  $i$  এর আগের মানটি ই শূন্য থেকে বড় কিনা যাচাই করা হচ্ছে। কিন্তু যখন প্রিন্ট করা হচ্ছে তখন  $i$  এর মান এক কমে গিয়েছে। তাই আমাদের প্রিন্ট স্টেটমেন্ট এ  $i+1$  দিতে হয়েছে।

## do-while loop

এক থেকে দশ পর্যন্ত সংখ্যা গুলো এবার do-while লুপ দিয়ে প্রিন্ট করব।

### **booktest.java**

```
public class booktest {
    public static void main(String[] args) {
        int i = 1;
        do {
            System.out.println(i);
            i++;
        }while (i<=10);
    }
}
```

এই লুপ এর বেলায় আগে প্রথম লুপে ঢুকে তার পর শর্ত চেক করে। তাই do-while loop কমপক্ষে একবার ঘুরবেই।

## for-each loop

for-each loop সাধারণত Array, ArrayList এসবের বেলায় ব্যবহার করা হয়। Array, ArrayList এর অধ্যায়ে for-each loop নিয়ে আলোচনা করব।

## break and continue

Break ব্যবহার করা হয় একটি লুপকে পুরপুরি থামিয়ে দিতে। আর continue ব্যবহার করা হয় একটি iteration/step কে skip করতে।

```
for (int i = 0; i < 10; i++) {
    if (i==3){
        continue;
    }
    if (i==7){
        break;
    }
    System.out.print(i+" ");
}
```

এখানে লুপটি  $i=3$  এ স্কিপ করে যাচ্ছে লুপ। আর  $i=7$  এ লুপটি একেবারেই থেমে যাচ্ছে। এই লুপটির আউটপুট হবে।

0 1 2 4 5 6

## Label in loop

`break` এর সাহায্যে আমরা লুপ থামাতে পারি। তবে ব্রেক হয় যে লুপের ভেতরে `break` আছে সেইটাই। `Nested` লুপ এর বেলায় যদি একেবারে বাহিরের লুপ বা একদম ভিতরের লুপ ছাড়া অন্য যেকোনো লুপ থামাতে হয় তাহলে কিন্তু আমরা সাধারণ `break` দিয়ে তা করতে পারি না।

```
first: for (int i = 0; i < 10; i++) {
    second: for (int j = 0; j < 10; j++) {
        third: for (int k = 0; k < 10; k++) {
            if (k==5){
                break second;
            }
            if (k==8){
                break first;
            }
            System.out.println(i);
        }
    }
}
```

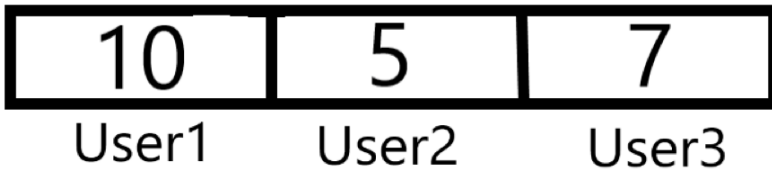
এখানে break second এবং break first যথাক্রমে first এবং second লুপকে থামিয়ে দিবে।

একই কথা continue এর জন্য ও প্রযোজ্য।

```
first: for (int i = 0; i < 10; i++) {
    second: for (int j = 0; j < 10; j++) {
        third: for (int k = 0; k < 10; k++) {
            if (k==5){
                continue first;
            }
            if (k==8){
                continue first;
            }
            System.out.println(i);
        }
    }
}
```

# অ্যারে

Array বহুল ব্যবহৃত একটি ডাটা টাইপ। এটি একটি রেফারেন্স টাইপ ডাটা টাইপ। Array তে আমরা একই টাইপের অনেকগুলো ডাটা রাখতে পারি। যেমন ধরি ইউজার ১ এর কাছে দশটি কলম আছে, ইউজার ২ এর কাছে আছে ৫ টি, ইউজার ৩ এর কাছে আছে ৭ টি। তিন জন ইউজারের ডাটা আমরা একটি array তে রাখতে পারি।



ডাটা গুলোর ইনডেক্সিং ও আছে। ইনডেক্স এর সাহায্যে আমরা ডাটা গুলো পেতে পারি। যেমনঃ ধরি Array টির নাম User। User array এর ০ ইনডেক্স এ আমরা ইউজার ১ এর ডাটা পাবো। অর্থাৎ ১০। এভাবে ইনডেক্স ১ এ ৫ এবং ইনডেক্স ২ তে ৭ পাবো।  
উল্লেখ্য, ইনডেক্সিং শুরু হয় শূন্য থেকে।  
এখন কোড এ চলে যাই।

একটি Array এর ডিক্লেয়ারেশন এরকমঃ

```
Data-type[] Array-name = new Data-type[Array-size];
```

Data-type যেকোনো তাই হতে পারে। int, float, double, Integer, Float, Double etc। রেফারেন্স টাইপ, প্রিমিটিভ টাইপ যেকোনো টাইপ ই হতে পারে। হতে পারে User-Defined data-type অর্থাৎ class বা interface। int type এর একটি array declaration দেখি।

```
int[] arr = new int[5];
```

এভাবে একটি array declare করতে হয়।

এবার দেখি array তে ডাটা রাখতে হয় কিভাবে।

আমি যেই array টি নিয়েছি তার সাইজ হচ্ছে ৫। সুতরাং array টিতে ইনডেক্সিং আছে ০ থেকে ৪ পর্যন্ত। যেহেতু ইনডেক্সিং শুরু হয় শূন্য থেকে। এবার array এর বিভিন্ন ইনডেক্স এ ডাটা রাখি।

```
arr[0] = 5;
arr[1] = 6;
arr[2] = 7;
arr[3] = 8;
arr[4] = 9;
```

এভাবে array টির বিভিন্ন ইনডেক্স এ ডাটা রাখতে পারি। array এর ডাটা টাইপ int হওয়াতে আমি = চিহ্নের ডান পাশে ইন্টিজার রেখেছি।

এবার দেখি ইনসার্ট করা ডাটা গুলো কিভাবে পাওয়া যায়। এর জন্য একটি for লুপ চালাতে পারি।

```
for (int i = 0; i < 5; i++) {
    System.out.println(arr[i]+" ");
}
```

সম্পূর্ণ কোডটি একেবারে দেখব এবার।

**Array.java**

```

public class Array {
    public static void main(String[] args) {
        int[] arr = new int[5];
        arr[0] = 5;
        arr[1] = 6;
        arr[2] = 7;
        arr[3] = 8;
        arr[4] = 9;
        for (int i = 0; i < 5; i++) {
            System.out.println(arr[i]+" ");
        }
    }
}

```

এতক্ষণ আমরা one dimensional array দেখলাম। এর পর দেখব multi dimensional array। Array one, two, three, four ... other dimensional হতে পারে।

এখন two dimensional array দেখি।

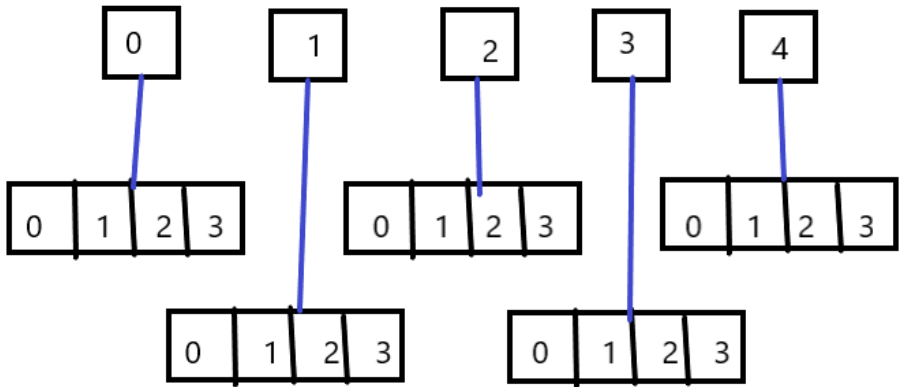
Two dimensional array এর declaration এরকম।

```

public class booktest {
    public static void main(String[] args) {
        int[][] twodarr = new int[5][4];
    }
}

```

নিচের চিত্রিত দেখলে 2-D array আর স্পষ্ট হবে। চিত্রে দেখা যাচ্ছে, পাঁচটি ঘরের প্রতিটি ঘরের জন্য চারটি করে ঘর রয়েছে।



নিচের উপায়ে ডাটা রাখতে পারি এবং উদ্ধার ও করতে পারি।

```
public class booktest {
    public static void main(String[] args) {
        int[][] twodarr = new int[5][4];

        twodarr[0][0] = 5;
        twodarr[0][1] = 4;

        System.out.println(twodarr[0][0]);
    }
}
```



যেহেতু অনেক বড় array। তাই আমরা for লুপ চালিয়ে data insert করব।

```
public class booktest {
    public static void main(String[] args) {
        int[][] twodarr = new int[5][4];

        //data insertion
        for (int i = 0; i < 5; i++) {
            for (int j = 0; j < 4; j++) {
                twodarr[i][j]=i+j;
            }
        }

        //data retrieving
        for (int i = 0; i < 5; i++) {
            for (int j = 0; j < 4; j++) {
                System.out.print(twodarr[i][j]+" ");
            }
            System.out.println();
        }
    }
}
```

আমরা চাইলে array এর সাইজ পরেও দিতে পারতাম।

```
public class booktest {
```

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

```
    int[][] twodarr = new int[5][];
```

twodarr[0] = new int[3]; //এখানে [0] হচ্ছে index আর [3] এটি হচ্ছে অই index এ যে array টি // রাখলাম তার সাইজ

```
    twodarr[1] = new int[4];
```

```
    twodarr[2] = new int[5];
```

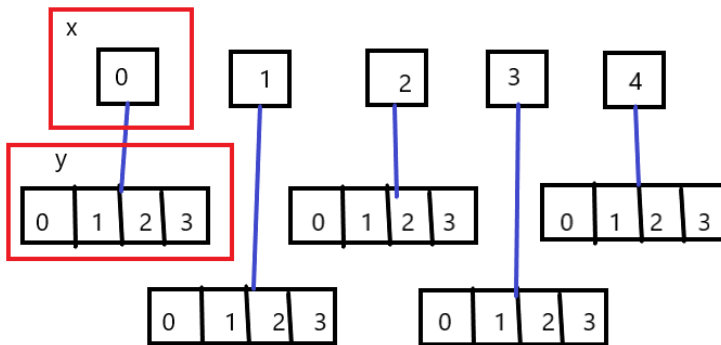
```
    twodarr[3] = new int[6];
```

```
    twodarr[4] = new int[7];
```

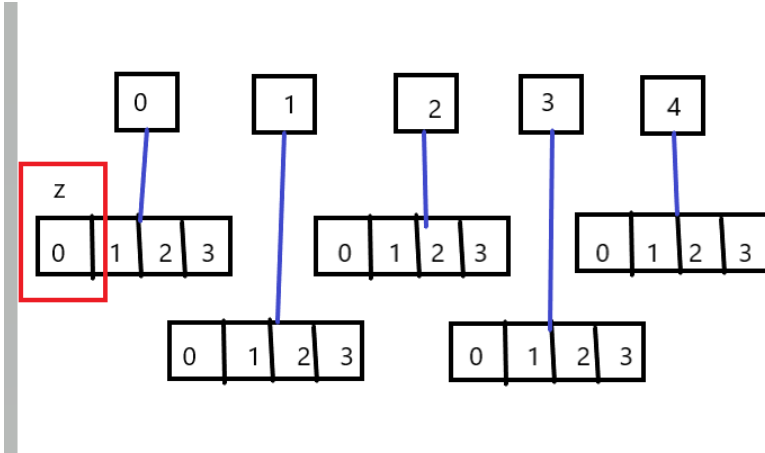
```
}
```

```
}
```

এখানে আসলে যা ঘটছে তা হল। twodarr এর [n] index পর্যন্ত যাওয়ার পর আমরা কিন্তু একটি single dimensional array কে assign করতে পারি অই index এ। আর [n][n] index পর্যন্ত গেলে আমরা একটি int assign করতে পারি।



চিত্রে যদি আমি x অবস্থানে আসি তবে আমি y অবস্থানে যা আছে অর্থাৎ একটি single dimensional array কে বসাতে পারি।



কিন্তু z অবস্থানে একটি int রাখতে পারি। এর কারন হচ্ছে এটি একটি two dimensional array তাই এর পর আর জায়গা নেই।

এবার আসি three dimensional array তে।

3-D array এর declaration এবং data insertion নিচে দেওয়া হল ।

```
public class booktest {
    public static void main(String[] args) {
        int[][][] threedarr = new int[5][4][3];
        threedarr[0][0][0] = 1;
        threedarr[0][0][1] = 2;
        threedarr[1][0][0] = 3;
        threedarr[2][0][0] = 4;
```

```
//many more
}
}
```

যেহেতু array টি বেশ বড়। তাই loop ব্যবহার করে data insert করি।

```
public class booktest {
    public static void main(String[] args) {
        int[][][] threedarr = new int[5][4][3];
        for (int i = 0; i < 5; i++) {
            for (int j = 0; j < 4; j++) {
                for (int k = 0; k < 3; k++) {
                    threedarr[i][j][k] = i+j+k;
                }
            }
        }
    }
}
```

3-D array তেও আমরা পরে সাইজ declare করতে পারি। ধরি একটি 3-D array নিম্নরূপ।

```
int[][][] arr = new int[5][][];
```

এখন আমরা যদি arr[n] এই index এ যাই তাহলে একটি 2-D array রাখতে পারব। যদি arr[n][n] এই index পর্যন্ত যাই তাহলে একটি single dimensional array রাখতে

পারব। আর যদি `arr[n][n][n]` এই index পর্যন্ত যাই তবে একটি int রাখতে পারব। নিচে উদাহরন দেওয়া হল।

`arr[3] = new int[2][3];` এখানে আমরা সাইজ 3 না দিয়ে ওই index এ পরবর্তীতে সাইজ দিতে পারতাম।

যেমন: `arr[3] = new int[2][];`

`arr[4][2] = new int[5];` এখানে 4,2 index 5 সাইজের একটি array রাখলাম।

`arr[2][1][0] = 5;` এখানে একটি int value রাখলাম।

## for-each loop এর সাহায্যে array প্রিন্ট করা

পূর্ববর্তী অধ্যায়ে বলেছিলাম Array তে এই লুপ নিয়ে আলোচনা করব। তো দেখে ফেলি কিভাবে for-each loop কাজ করে।

### **booktest.java**

```
public class booktest {
    public static void main(String[] args) {
        String[] arr = new String[5];
        arr[0] = "Abduz Zami";
        arr[1] = "Abdus Sami";
        arr[2] = "Manoara Begum";
        arr[3] = "Md Hazrat Ali";
```

```
arr[4] = "Tania Akter";
```

```
for (String name:
    arr) {
    System.out.println(name);
}
}
```

এখানে arr এর প্রতিটি উপাদান name এ আসে এবং আমরা name কে প্রিন্ট করি। অর্থাৎ array টির উপাদান গুলোর যেই ডাটা টাইপ, সেই ডাটা টাইপের একটি চলকে arr[0],arr[1],.... এমন অন্য ডাটা গুলোকে একের পর এক রাখা হয়। আর আমরা তাকে প্রিন্ট করতে পারি।

# মেথড

মেথড জিনিসটা অনেকের কাছেই নতুন লাগতে পারে। তবে আমরা সবাই ই হয়ত ফাংশন এর সাথে পরিচিত। জাভায় ফাংশনকেই মেথড বলা হয়। একটি ফাংশনের চারটি উপাদান থাকে-

- ১। ইনপুট টাইপ
- ২। রিটার্ন টাইপ
- ৩। মেথড এর নাম
- ৪। বডি

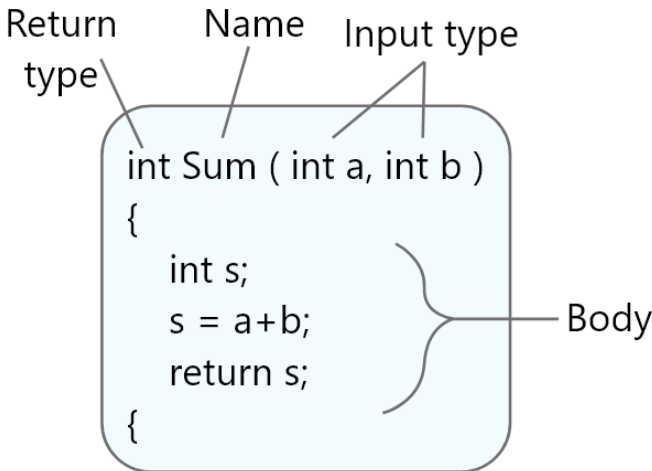


Figure: Method Prototype

চিত্রে একেবারে সাধারণ একটি ফাংশন বা মেথড দেখানো হয়েছে। এখন মেথডের উপাদান গুলো নিয়ে কিছু কথা বলি।

## ইনপুট টাইপ

মেথডটি কোন ধরনের ডাটা ইনপুট নিবে তাই এখানে বলে দেওয়া হয়। উপরের উদাহরণে মেথডের ইনপুট টাইপ হচ্ছে ইন্টিজার।

## রিটার্ন টাইপ

মেথড এর কাজ শেষে মেথডটি থেকে কোন ধরনের ডাটা পাওয়া যাবে তাই এখানে বলা থাকে।

## মেথড এর নাম

এখানে মেথডটির একটি নাম দেওয়া হয়।

## বডি

এখানে মেথডটি কি কাজ করবে এবং কাজ শেষে কি রিটার্ন করবে তা বলা থাকে।

এখন খুব সহজ একটি কোড লিখি।

### test1.java

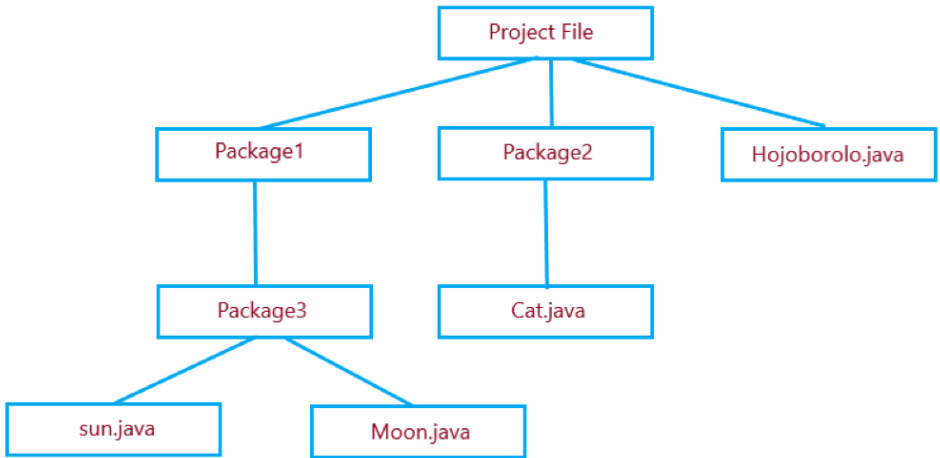
```
public class test1 {
    static int sum(int a, int b){
        int s;
        s=a+b;
        return s;
    }
    public static void main(String[] args) {
        int s = sum(5,10);
        System.out.println(s);
    }
}
```

উপরের উদাহরণে যা করেছিলাম এখানেও তাই করেছি। এখানে একটি নতুন জিনিস দেখা যাচ্ছে static। প্রশ্ন হচ্ছে static কি? Static নিয়ে আমরা পরে জানব। এখন শুধু এইটুকু বুঝি যে static মেথড থেকে non-static মেথড কে কল করা যায় না। যেহেতু main মেথড একটি static মেথড, তাই sum মেথডকে main মেথড থেকে কল করতে sum কেও static করতে হয়েছে।



স্ট্রিং

# জাভা প্রজেক্ট এর গঠন



এটা একটি জাভা প্রজেক্ট এর গঠন হতে পারে। প্রজেক্ট ফাইলের ভিতর অনেক প্যাকেজ থাকতে পারে। জাভা ফাইল থাকতে পারে। প্যাকেজ এর ভিতরে প্যাকেজ ও থাকতে পারে। জাভা ফাইল গুলোর মধ্যে একটি জাভা ফাইলে অবশ্যই main মেথড থাকতে হবে। অর্থাৎ উপরের উদাহরণ গুলোর মধ্যে Hojoborolo, Cat, sun, Moon এদের যেকোনো একটির ভিতরে অবশ্যই main মেথড থাকতে হবে। নতুবা প্রজেক্ট রান হবে না।

# প্যাকেজ

প্যাকেজ এর কাজ হচ্ছে কোডের readability বাড়ানো। অনেক সময় একটি প্রোজেক্ট একাধিক কোডার এর সমন্বয়ে করতে হয়। একটি প্যাকেজের ভিতর যদি মোটামুটি এক ক্যাটাগরির ক্লাস গুলো রাখা হয় বা যদি কত গুলো ক্লাস একই কাজের জন্য ব্যবহার হয় সেক্ষেত্রে ওই ক্লাস গুলোকে একই প্যাকেজের ভিতরে রাখলে পরবর্তীতে আপগ্রেড করতে বা বাগ ফিক্স করতে সুবিধা হয়।

প্যাকেজের ভিতরে যদি কোন ক্লাস তৈরি করা হয় তবে একদম উপরে package এর নাম লিখে দিতে হয়। যেমন যদি Cat ক্লাস টি package2 এর অন্তর্ভুক্ত হয় তবে কোডটি হবে নিম্নরূপ।

## Cat.java

package package2; //এই লাইনটির অর্থ হল এই যে এটি package2 এর অন্তর্ভুক্ত

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

    }
}
```

যদি এমন হতো।

## Cat.java

`package package1.package2;` //এই লাইনটির অর্থ হল এই যে এটি `package1` এর `package2` এর অন্তর্ভুক্ত অর্থাৎ একটি প্যাকেজ এর ভিতরে অন্য একটি প্যাকেজ। এভাবে nested প্যাকেজ ও তৈরি করা যায়।

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

# ক্লাস এবং অবজেক্ট

ক্লাস আর অবজেক্ট নিয়ে কিছু বাস্তব উদাহরণ দিয়ে বোঝানোর চেষ্টা করি।

Animal যদি class হয়। মানুষ, গরু, ছাগল, হাঁস, মুরগি, কুকুর, বিড়াল সব হল animal ক্লাসের object। Object গুলোর প্রত্যেকের কিন্তু কিছু similar বৈশিষ্ট্য আছে আবার কিছু different বৈশিষ্ট্য আছে।

Java তে প্রত্যেকটি java ফাইল ই এক একটা class. অর্থাৎ প্রত্যেক জাভা ফাইলে একটি class থাকতেই হবে যার নাম আর ফাইলের নাম একই হবে। একাধিক class ও থোলা যায়। এক্ষেত্রে Main ক্লাসটি অবশ্যই public হবে। বাকি class গুলো public করা যাবেনা।

একটি প্রোজেক্ট এ এক বা একাধিক জাভা ফাইল থাকতে পারে। তার মধ্যে একটি জাভা ফাইলে অবশ্যই একটি main মেথড থাকতে হবে। FUNCTION কে জাভায় METHOD বলে। এক ফাইলে অনেক জাভা ক্লাস থোলা গেলেও ভিন্ন ভিন্ন ফাইলে ক্লাস থোলাই শ্রেয়।

এখন আমরা উদাহরণ দেখি।

আমরা Main.java নামে একটি জাভা ফাইল খুলি।

Main.java ফাইলের ভিতরে আমাদের একটি class নিতে হবে যার নাম Main হবে। অর্থাৎ ফাইল এর নাম এবং ক্লাস এর নাম একই হতে হবে। আরেকটি কাজ করতে হবে যা হচ্ছে একটি main method নিতে হবে। সহজ কথায়, যে ফাইলটি আমরা রান করব সেটি তে একটি main method থাকতে হবে।

**Main.java**

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

    }
}
```

এখন Main.java ফাইলেই ClassOne নামে একটি ক্লাস খুলি

**Main.java**

```
class ClassOne{
    int x;
}

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

    }
}
```

আমরা ClassOne.java নামে একটি জাভা ফাইল খুলে সেখানেও ClassOne ক্লাসটি declare করতে পারতাম।

**Main.java**

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

    }
}
```

### **ClassOne.java**

```
class ClassOne{
    int x;
}
```

ClassOne এ main method এর প্রয়োজন নেই। কারণ আমরা ClassOne.java ফাইল কে রান করব না। রান করব শুধু Main.java ফাইল। Main.java ফাইল থেকে ClassOne.java ফাইল কল হবে। কল করার জন্য আমাদের Main class এ ClassOne এর অবজেক্ট তৈরি করতে হবে। আমরা পরবর্তীতে দেখব কিভাবে অবজেক্ট তৈরি করতে হয়।

এর মানে এই যে, কোন প্রজেক্ট রান করলে সবার আগে যে মেথড টি execute হবে তাকেই main method বলে। Main method অন্য জাভা ফাইলে থাকা ক্লাস ও মেথড গুলোকে পর্যায়ক্রমে প্রয়োজন অনুসারে কল করবে।

আরেকটা কথা বলে রাখি। JAVA IDE গুলোতে প্রোজেক্ট আকারে ওপেন করার পর অনেকের পরস্পর সম্পর্কহীন একাধিক জাভা ফাইল নিয়ে কাজ করতে হয়। বিশেষ করে যারা competitive programming এর সাথে জরিত। এক্ষেত্রে IDE গুলোতে single file

হিসেবে রান করার option থাকে। এরকম কাজ করতে গেলে অবশ্যই প্রতিটি জাভা ফাইলে ক্লাসের ভিতরে একটি main method নিতেই হবে।

এখন আমরা অবজেক্ট তৈরি করব

## Main.java

```
public class Main {
    public static void main(String[] args){
        ClassOne classone = new ClassOne();
    }
}
```

এভাবে আমরা ClassOne এর একটি অবজেক্ট তৈরি করতে পারি। এখানে classone হচ্ছে reference variable। আগে বলেছিলাম class হচ্ছে একটি reference type data type। new দিয়ে ClassOne এর অবজেক্ট declare করা হয়েছে এবং মেমোরিতে এই অবজেক্ট এর জন্য কিছু জায়গা বরাদ্দ হয়েছে। Object declaration নিয়ে পরবর্তীতে আরও বিস্তারিত বলব। Memory allocation এ কিছু ধারণা দিচ্ছি।



# রেফারেন্স টাইপের মেমোরি বন্টন

আমরা যদি রেফারেন্স টাইপ এর ক্ষেত্রে মেমোরি allocation দেখতে চাই।

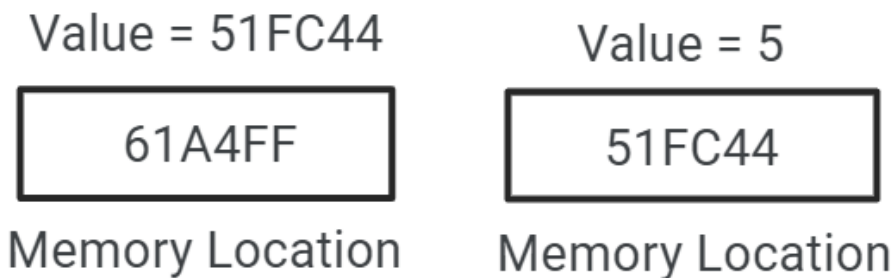
একটি নন-প্রিমিটিভ বা রেফারেন্স টাইপ variable ডিক্লেয়ার করি,

```
Flower x = new Flower(5);
```

এখন আমরা জানি কিভাবে একটি ক্লাসের অবজেক্ট declare করতে হয়।

এখানে x হচ্ছে Flower ক্লাসের এর একটি রেফারেন্স variable। new দিয়ে Flower ক্লাসের একটি অবজেক্ট তৈরি করা হয়েছে যেখানে 5 হচ্ছে Flower ক্লাসের কোন একটি int টাইপ variable এর value এবং এই অবজেক্ট টির জন্য একটি মেমোরি allocate করা হয়েছে। এবং এই allocated লোকেশন কে রেফার করা হয়েছে x এর মাধ্যমে।

তাহলে মেমরি তে এর representation হবে অনেকটা এরকম,



এই চিত্র লক্ষ্য করলে আমরা দেখতে পারবো যে 61A4FF এই মেমরি এড্রেস এ x এর মান যেই লোকেশন এ রাখা আছে সেই লোকেশন টি রাখা আছে। তার মানে প্রথম লোকেশন টি অন্য একটি লোকেশন কে রেফার করে। এই জন্য এই ধরনের ডাটা টাইপ কে আমরা বলছি রেফারেন্স টাইপ।

তবে প্রকৃতপক্ষে memory allocation আরও জটিল একটি বিষয়। একটি মেমোরি অ্যাড্রেস এ একটি int কে রাখা সম্ভব নয়। কারণ একটি মেমোরি অ্যাড্রেস এ ১ বাইট জায়গা থাকে। int যেহেতু ৪ বাইট জায়গা দখল করে সেহেতু int এর 4 টি মেমোরি অ্যাড্রেস লাগবে। যে অ্যাড্রেস টি আমি উদাহরণ স্বরূপ দিয়েছি সেটিকে আমরা starting address বলতে পারি। রেফারেন্স টাইপ বোঝানোর জন্য এই উদাহরণ টি দেওয়া।

এবার আরও বড় একটি উদাহরণ দেই,  
একটি ক্লাস তৈরি করি,

```
Class Vehicle{
    Int wheels;
    Int windows;
    Int weight;

    Vehicle(){} // Constructor
}
```

Constructor এর সাহায্যেই অবজেক্ট তৈরি করতে হয়। Constructor নিয়ে একটু পরেই বিস্তারিত আলোচনা করব।

ক্লাসটি তৈরি করার পর main ক্লাসের main মেথড এ একটি অবজেক্ট তৈরি করি।

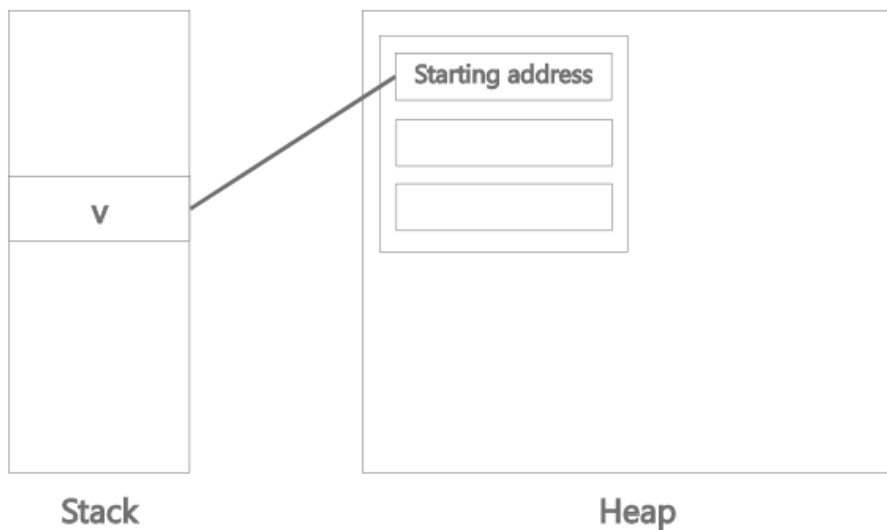
```
public class Main {

    public static void main(String[] args) {
        Vehicle v = new Vehicle();
    }
```

}

নিচের চিত্রটি লক্ষ্য করি।

এখানে দুটো জিনিস দেখতে পারছি। একটি Stack memory অন্যটি Heap memory.



স্ট্যাক স্পেস মূলত method execution এবং local ভেরিয়েবলের ক্রম সংরক্ষণের জন্য ব্যবহৃত হয়। এটি সর্বদা স্টোর কৃত ব্লকগুলি LIFO (Last in first out) ক্রমে স্ট্যাক করে।

অন্যদিকে হিপ মেমরি dynamic memory allocation ব্যবহার করে মেমরি ব্লকগুলি বরাদ্দকরণ এবং ডিলিট করার জন্য নিয়োজিত।

এখন যদি Vehicle ক্লাসের অবজেক্ট দিয়ে চিত্রটি বোঝাতে চাই,

এখানে `v` হচ্ছে `Vehicle` এর একটি রেফারেন্স `variable` যা `Vehicle` এর অবজেক্ট কে রেফার করছে।

`Vehicle` class এর object এর জন্য কি পরিমাণ মেমোরি দরকার তা `heap` মেমরিতে `allocate` করে সেই মেমোরি ব্লক এর প্রথম অ্যাড্রেসটি `stack` স্পেস এ অবস্থানরত `v` এর কাছে সংরক্ষিত থাকে।

এসব না জানলেও কোডিং করতে সমস্যা হবে না। তবে ধারণা থাকা ভাল।

# কন্সট্রাক্টর

Constructor এর সাহায্যে কোন একটি ক্লাসের অবজেক্ট তৈরি করা হয়।

আমরা Animal নামে একটি ক্লাস খুলি। Animal.java নামে একটি জাভা ফাইল তৈরি করে Animal class তৈরি করি।

## Animal.java

```
public class Animal {
    String name;
    int legs;
    boolean tail;

    void displayinformation()
    {
        System.out.println("name : "+name+"\n"+"legs : "+legs+"\n"+"tails : "+tail);
    }

    Animal(String iname, int ilegs, boolean itail) {
        name = iname;
        legs = ilegs;
        tail = itail;
    }
}
```

Animal class টিতে name এর জন্য একটি String, কতগুলি পা আছে টা রাখার জন্য int type একটি variable legs এবং লেজ আছে কি নেই তা রাখার জন্য boolean type একটি variable tail নিয়েছি।

এরপর একটি মেথড নিয়েছি displayinformation, যার return type void যেহেতু মেথড টি কোন কিছু return করবেনা। এই মেথড দিয়ে আমরা Animal class এর কোন অবজেক্ট এর ইনফর্মেশন গুলো প্রিন্ট করব।

তারপর constructor declare করেছি। Constructor একটি মেথড যার কাজ হচ্ছে অবজেক্ট তৈরি করা, অবজেক্ট এর value initialize করা। Constructor এর নাম আর ক্লাসের নাম এক এ হতে হবে। এর কোন রিটার্ন টাইপ নেই। এমনকি void ও দিতে হবেনা। Constructor এ parameter হিসেবে variable এর value গুলো নিয়ে সেগুলো কে নির্দিষ্ট অবজেক্ট এর জন্য assign করে দিতে হয়। যেমনঃ parameter হিসেবে iname নিয়ে সেটিকে object এর name variable এ assign করা হয়েছে।

এখানে বলে রাখি parameter এ name নিয়ে object এর name এ assign করা যেত। তবে তার জন্য আমাদের this keyword ব্যবহার করতে হবে। আমরা যখন this keyword এর ব্যবহার শিখব তখন এভাবে করে দেখব।

এখন Main.java নামে একটি জাভা ফাইল খুলে তাতে Main নামে একটু ক্লাস খুলি

## Main.java

```
public class Main {
    public static void main(String[] args) {
        Animal dog= new Animal("Dog",4,true);
```

```
}
}
```

এভাবে আমরা Animal class এর একটি অবজেক্ট dog তৈরি করে ফেললাম। আমরা চাইলে displayinformation মেথড এর সাহায্যে dog এর ইনফর্মেশন গুলো দেখতে পারি।

```
public class Main {
    public static void main(String[] args) {
        Animal dog = new Animal("Dog",4,true);
        dog.displayinformation();
    }
}
```

এভাবে অবজেক্ট এর সাহায্যে কোন ক্লাসের মেথড কল করতে হয়। Output এ আমরা যা দেখতে পারব।

```
name : Dog
legs : 4
tails : true
```

আরও একটি অবজেক্ট তৈরি করি। এবং displayinformation মেথডের সাহায্যে ইনফর্মেশন গুলো দেখি।

## Main.java

```
public class Main {
```

```

public static void main(String[] args) {
    Animal dog = new Animal("Dog",4,true);
    dog.displayinformation();
    Animal human = new Animal("Human",2,false);
    human.displayinformation();
}
}

```

Output এ যা দেখা যাবে

```

name : Dog
legs : 4
tails : true
name : Human
legs : 2
tails : false

```

সব গুলো variable initialize না নিয়েও constructor তৈরি করা যায়। আবার কোন variable initialize না করেও constructor তৈরি করা যায় যাকে default constructor বলে।

আমরা default constructor দেখব এখন

```

public class Animal {
    String name;
    int legs;

```



```
boolean tail;
```

```
void displayinformation()
```

```
{
```

```
    System.out.println("name : "+name+"\n"+"legs : "+legs+"\n"+"tails :
```

```
    "+tail);
```

```
}
```

```
Animal() {
```

```
}
```

```
}
```

এখানে Animal() হচ্ছে default constructor। আমরা যদি কোন constructor তৈরি না করি তাহলে Object তৈরি করতে গেলে default constructor তৈরি করতে হয়।

ধরে নেই আমরা কোন constructor তৈরি করিনি

```
public class Animal {
```

```
    String name;
```

```
    int legs;
```

```
    boolean tail;
```

```
    void displayinformation()
```

```
{
```

```
        System.out.println("name : "+name+"\n"+"legs : "+legs+"\n"+"tails :
```

```
        "+tail);
```

```
}
```

```
}
```

এখন Main ক্লাস থেকে যদি Animal class এর একটি অবজেক্ট তৈরি করতে চাই।

```
public class Main {
    public static void main(String[] args) {
        Animal dog= new Animal();
    }
}
```

এক্ষেত্রে default constructor কল হবে যেহেতু আমরা Animal class এ কোন constructor তৈরি করিনি। এক্ষেত্রে কোন value initialize হবেনা। আমাদের আলাদা আলাদা ভাবে তখন value initialize করতে হবে।

```
public class Main {
    public static void main(String[] args) {
        Animal dog = new Animal();
        dog.name="Dog";
        dog.legs=4;
        dog.tail=true;
        dog.displayinformation();
    }
}
```

আমরা কয়েকটি variable নিয়েও constructor তৈরি করতে পারি

```
public class Animal {
    String name;
```

```
int legs;
```

```
boolean tail;
```

```
void displayinformation()
```

```
{
```

```
    System.out.println("name : "+name+"\n"+"legs : "+legs+"\n"+"tails :
```

```
" +tail);
```

```
}
```

```
Animal(String iname, int ilegs, boolean itail) {
```

```
    name = iname;
```

```
    legs = ilegs;
```

```
    tail = itail;
```

```
}
```

```
public Animal(String iname, int ilegs) {
```

```
    name = iname;
```

```
    legs = ilegs;
```

```
}
```

```
public Animal(String iname) {
```

```
    name = iname;
```

```
}
```

```
Animal() {
```

```
}
```

```
}
```

এমনকি একাধিক constructor ও থাকতে পারে একটি ক্লাসে।

আমরা যদি default constructor ছাড়া বাকি constructor গুলো তৈরি করি তাহলে default constructor কিন্তু আর automatically call হবে না। সে ক্ষেত্রে default constructor call করতে হলে আমাদের অবশ্যই default constructor অর্থাৎ parameterless / argumentless constructor তৈরি করে নিতেই হবে।

This keyword এর সাহায্যে Constructor তৈরি করা আরও সুবিধাজনক।

```
public class Animal {
    String name;
    int legs;
    boolean tail;

    void displayinformation()
    {
        System.out.println("name : "+name+"\n"+"legs : "+legs+"\n"+"tails : "+tail);
    }

    public Animal(String name, int legs, boolean tail) {
        this.name = name;
        this.legs = legs;
        this.tail = tail;
    }
}
```

```
Animal() {  
    }  
}
```

এখানে `this.name` Animal class এর name কে নির্দেশ করে। আর = এর ডান পাশের 'name' Constructor এর parameter এর name নির্দেশ করে।

This keyword নিয়ে পরবর্তীতে বিস্তারিত আলোচনা করব।

# ফাইনলাইজ মেথড

জানায় কোন destructor নেই। finalize নামে একটি মেথড আছে যা অনেকটা destructor এর মত। তবে এটি destructor নয়। যখন মেমোরি সঙ্কট দেখা দেয় তখন finalize মেথডটি java virtual machine দ্বারা automatically কল হয়। finalize মেথড যে কাজটি করে - যেসব অবজেক্ট আর প্রয়োজন হবে না সেগুলোর মেমোরি ফাকা করে দেয়।

## testbook.java

```
class Hojoborolo{
    public Hojoborolo() {
        System.out.println("Object created at"+this);
    }

    @Override
    protected void finalize() {
        System.out.println("Finalized called at"+this);
    }
}

public class testbook {
    public static void main(String[] args) {
        while(true)
        {
            new Hojoborolo();
        }
    }
}
```

```

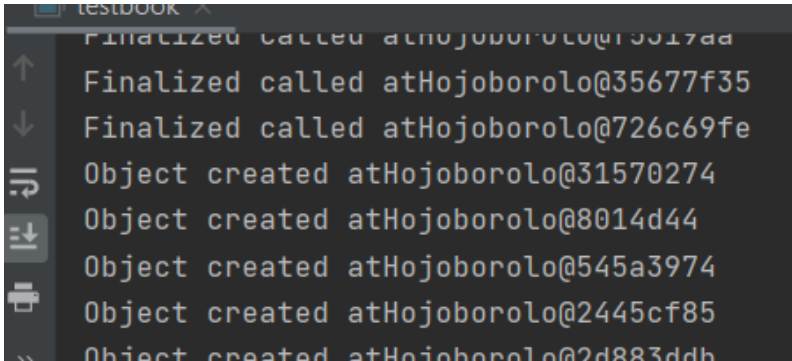
    }
}
}

```

এখানে একই ফাইলে দুটি ক্লাস তৈরি করেছি। আগে বলেছিলাম যে একই ক্লাস এ দুইটি ক্লাস তৈরি করা যায়। Hojoborolo নামে একটি ক্লাস তৈরি করেছি। তার মধ্যে কন্সট্রাক্টর তৈরি করেছি আর finalize মেথড কে ওভাররাইড করেছি। constructor আর finalize মেথড এর ভেতরে প্রিন্ট করেছি যাতে বোঝা যায় কখন কে কল হচ্ছে।

Testbook ক্লাস (এটি এই ফাইলের মেইন ক্লাস) এর ভিতরে একটি while লুপ infinite সংখ্যক বার চালিয়েছি যাতে আমরা অসংখ্য অবজেক্ট তৈরি করতে পারি।

এবার যদি কোডটি রান করি তাহলে দেখতে পাবো যে অবজেক্ট তৈরি হচ্ছে আর একটু পর পর finalize হচ্ছে।



```

testBOOK X
Finalized called atHojoborolo@3517aa
Finalized called atHojoborolo@35677f35
Finalized called atHojoborolo@726c69fe
Object created atHojoborolo@31570274
Object created atHojoborolo@8014d44
Object created atHojoborolo@545a3974
Object created atHojoborolo@2445cf85
Object created atHojoborolo@2d883ddb

```

উল্লেখ্য, এখানে this একটি রেফারেন্স দেয়। এটি কিন্তু মেমোরি অ্যাড্রেস নয়। জাভায় আমরা সরাসরি মেমোরি নিয়ে কাজ করতে পারিনা।

# প্রিমিটিভ টাইপ বনাম রেফারেন্স টাইপ

আমরা Main.java নামে একটি জাভা ফাইল তৈরি করি

## Main.java

```
public class Main {
    public static void main(String[] args) {
        int x = 10;
        int k = x;
    }
}
```

এটির memory allocation যদি দেখি আমরা দেখতে অনেকটা এরকম হবে।

a = 10
--------

606AFG

b = 10
--------

515AAA

অর্থাৎ a এবং b এর জন্য আলাদা আলাদা মেমোরি স্পেস প্রয়োজন হবে।



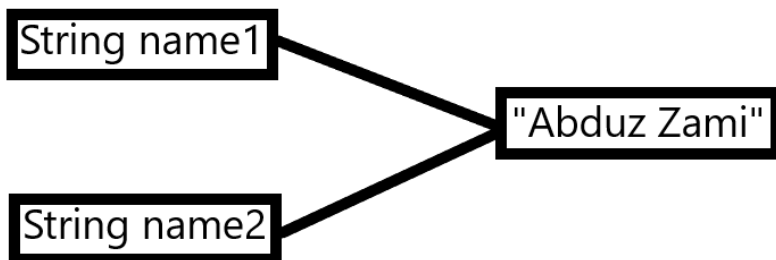
অন্যদিকে যদি আমরা Reference type এর মেমোরি এলোকেশন দেখি।

উপরের কোডটি কিছু পরিবর্তন করি

### Main.java

```
public class Main {
    public static void main(String[] args) {
        String name1 = new String("Abduz Zami");
        String name2 = name1;
    }
}
```

এই কোডটির memory allocation যদি দেখি আমরা তাহলে অনেকটা এরকম।



অর্থাৎ name1 আর name2 আলাদা আলাদা “Abduz Zami” তৈরি না করে একই “Abduz Zami” কে নির্দেশ করছে।

এর মানে এই যে এরা একই object কে refer করছে। মেমোরি তে name2 এর জন্য আলাদা object তৈরি হয়নি।

# কল বাই ভ্যালু

**testbook.java**

```
public class testbook {  
    static void change(int x)  
    {  
        x = 500;  
    }  
    public static void main(String[] args) {  
        int num = 10;  
        change(num);  
        System.out.println(num);  
    }  
}
```

Output:

10

এখানে change মেথডটিতে num কে পাঠানোর পরে মান পরিবর্তন করলেও num এর প্রকৃত মানের কিন্তু কোন পরিবর্তন হয়নি। এটিকেই বলা হয় কল বাই ভ্যালু। অর্থাৎ এখানে num এর ভ্যালু x এ কপি হয়েছিল। x এবং num এরা দুইজন এ আলাদা variable।

এটাকেই বলা হয় কল বাই ভ্যালু।

আর এখানে static ব্যবহার করার কারন হল static method থেকে শুধু মাত্র static মেথড কে কল করা যায়। main মেথড যেহেতু static তাই main মেথড থেকে change মেথড কে কল করতে change মেথড কেও static করতে হয়েছে।

# কল বাই রেফারেন্স

**testbook.java**

```
class Hojoborolo{  
    int age;  
}  
  
public class testbook {  
    static void change(Hojoborolo h)  
    {  
        h.age = 400;  
    }  
    public static void main(String[] args) {  
        Hojoborolo h1 = new Hojoborolo();  
        h1.age = 50;  
        change(h1);  
        System.out.println(h1.age);  
    }  
}
```

Output:

400

এখানে Hojoborolo একটি ক্লাস তৈরি করেছি। এই ক্লাসের একটি variable হচ্ছে age। testbook ক্লাসে change মেথড তৈরি করেছি যা Hojoborolo ক্লাসের রেফারেন্স

variable গ্রহন করে এবং age এর মান পরিবর্তন করে দেয়। main মেথড এ Hojoborolo ক্লাসের অবজেক্ট তৈরি করেছি h1। h1 এর age এর একটি মান দিয়েছি। h1 কে change মেথড এ পাঠিয়েছি। এর পর output এ দেখতে পারছি যে age এর মান পরিবর্তিত হয়েছে। এটাই হচ্ছে কল বাই রেফারেন্স।

উল্লেখ্য, এখানে h1 কিন্তু রেফারেন্স variable। এখানে দুইটি রেফারেন্স variable h1 এবং h আসলে একই অবজেক্ট কে নির্দেশ করছে। তাই একটি তে পরিবর্তন করলেই অন্যটিতেও পরিবর্তন হয়ে যাচ্ছে।

# রেপার ক্লাস

Wrapper class যে কাজ টি করে টা হল primitive type কে একটি অবজেক্ট এর ভিতরে রেখে দেয়।

Wrapper class এর অবজেক্ট ডিক্লেয়ার করার সাধারণ নিয়ম:

```
Type variable = Type(value);
```

তবে এদেরকে primitive টাইপের মত করেও ডিক্লেয়ার করা যায়। বার বার ডিক্লেয়ার করা হয় বলে প্রিমিটিভ টাইপের মত ডিক্লেয়ার এর সুযোগ দিয়েছে জাভা। যেমন :

```
Type variable = value;
```

Wrapper class যে কাজ টি করে টা হল primitive type কে একটি অবজেক্ট এর ভিতরে রেখে দেয়।

Byte:

```
Byte b = new Byte(127);
```

এভাবে ডিক্লেয়ার করলে হয়ে যাওয়ার কথা তাই না? কিন্তু হবে না। কারন এখানে 127 কে কম্পাইলার int হিসেবে ধরে নিচ্ছে। এজন্য 127 কে type casting করতে হবে। নিচে টাইপ কাস্টিং করে আগের কোডটি লিখা হল।

```
Byte b = new Byte((byte) 127);
```

টাইপ কাস্টিং নিয়ে কিছু কথা বলি।

টাইপ কাস্টিং এ এক টাইপের ডাটা কে অন্য টাইপে কনভার্ট করা হয়। যে টাইপে নিতে হবে সেই টাইপকে প্রথম বন্ধনীর ভিতরে উল্লেখ করতে হবে। তবে এক্ষেত্রে দুটো ডাটার সাইজ এক হতে হবে। যেমন এখানে 127 কে Byte এ নিতে পারছি কারন 127 এর যা সাইজ তা Byte ডাটা টাইপের সাইজ 1 byte এর সমান। সহজ কথায় = এর ডান পাশে যা দিব তা যে টাইপে cast করব তার range এর ভিতরে হতে হবে। যেমন নিচের উদাহরণটি যদি দেখি।

```
double e = 10.012;
int h = (int) e;
System.out.println(h);
```

এখানে double e কে int এ cast করা সম্ভব হয়েছে। কারন 10 int এর range -2,147,483,648 থেকে 2,147,483,647 এর ভিতরে রয়েছে। যদি এমন করি

```
double e = 1000000000000000.122222222;
int h = (int) e;
System.out.println(h);
```

এক্ষেত্রে output পাব তবে ভুল output। এখানে, = এর ডান পাশের সংখ্যার সাইজ int এর maximum সাইজ কে অতিক্রম করে ফেলেছে। তাই ভুল আউটপুট দেখাচ্ছে।

অনেক ক্ষেত্রে এই type casting গুলো automatically হয়ে যায়। একে বলে implicit type casting।

আমরা কিন্তু উপরের Byte এর declaration এভাবেও করতে পারতাম।

```
Byte b = 127;
```

এখানে implicit type casting হয়ে যেত।

কিছু ডাটা টাইপ প্রচুর ব্যবহার হয় বলে জাভায় Byte এর মত কয়েকটা ডাটা টাইপ কে primitive টাইপের মত করে declare করার সুযোগ দিয়েছে। Boolean, Byte, Integer, Short, Long, Double, Float, Char, String এগুলোকে primitive টাইপ এর মত করে declare করা যায়।

আরেকটি গুরুত্বপূর্ণ কথা বলি, জাভায় double x = 101.23 declare করার মানে এই না যে = এর ডান পাশে যা দেওয়া হবে তাকে double হিসেবে ধরা হবে। এমনটা নয়। জাভা কম্পাইলার ভগ্নাংশ পেলেই double হিসেবে ধরে নিবে। কিছু ক্ষেত্রে implicit casting হবে। short s = 10000 এখানে কম্পাইলার 10000 কে integer হিসেবেই ধরে নিচ্ছে, কিন্তু 10000 short এর range এর মধ্যে থাকায় type casting হয়ে যাচ্ছে। এখানে আমি এতাই বোঝাতে চেয়েছি যে জাভা কম্পাইলার ডাটা টাইপ দেখে ধরে নেয় না যে = এর ডান পাশে কি থাকবে।

**Integer:**

এবার আসি Integer এ।

```
Integer x = new Integer(5);
```

এখানে টাইপ কাস্টিং করার প্রয়োজন হয়নি। কারন 5 কে কম্পাইলার Integer হিসেবেই ধরে নিয়েছে।

এভাবে ডিক্লেয়ার করা যায়। তবে শর্টকাট আছে। যেহেতু Integer বার বার ডিক্লেয়ার করার দরকার পরে। তাই জাভায় এটিকেও প্রিমিটিভ টাইপের মত করে ডিক্লেয়ার করার সুযোগ দিয়েছে।

এভাবে ডিক্লেয়ার করতে পারি আমরা

```
Integer x = 5;
```

**Short:**

```
Short x = new Short((short) 5);
```

এভাবে ডিক্লেয়ার করলে টাইপ কাস্টিং করে নিতে হবে।



অত ঝামেলার দরকার নেই। নিচের মত করে ডিক্লেয়ার করব।

Short x = 5;

Long:

Long lo = new Long(1000000000000000000L);

এভাবে ডিক্লেয়ার করা যায় তবে নিচের পদ্ধতিতে করাই বুদ্ধিমানের কাজ।

Long x = 1000000000000000000L;

Double:

Double x = 1001.2345;

Float:

Float f = 1012.345F;

Char:

Char c = 'A';

Boolean:

Boolean b = false;

Double থেকে Boolean পর্যন্ত short declaration ই দেখালাম শুধু।

এতক্ষণ আমরা দেখলাম কিভাবে একটি প্রিমিটিভ টাইপ কে অবজেক্ট এ নিতে পারি wrapper class এর মাধ্যমে।

# ইনহেরিটেন্স

অবজেক্ট ওরিয়েন্টেড প্রোগ্রামিং এর অন্যতম গুরুত্বপূর্ণ বিষয় হচ্ছে ইনহেরিটেন্স।

Inheritance এর বাংলা হচ্ছে উত্তরাধিকার। সন্তান-সন্ততি যেমন বাবা মার সম্পত্তি উত্তরাধিকার সূত্রে পায়, তেমনি জাভার একটি ক্লাস ও অন্য ক্লাসের প্রোপার্টি ইনহেরিট করতে পারে। আমরা কোড এর মাধ্যমে বোঝার চেষ্টা করি।

## Animal.java

```
package package1;
```

```
public class Animal {
    String name;
    int age;
}
```

## Dog.java

```
package package1;
```

```
public class Dog {
    String name;
    int age;
    int legs;
}
```

উপরে দুইটি ক্লাস তৈরি করেছি। একটি Animal class অন্যটি Dog class। Dog class এর দিকে যদি লক্ষ্য করি তাহলে দেখতে পারব যে, এর দুটি variable name এবং age কিন্তু Animal class এও ছিল। এখন আমরা যদি Dog ক্লাস থেকে Animal ক্লাস কে ইনহেরিট করতাম তাহলে Animal ক্লাসের ওই দুটি variable Dog ক্লাসেও চলে আসতো। inherit করার জন্য আমাদের extend keyword ব্যবহার করতে হবে।

## Dog.java

```
package package1;

public class Dog extends Animal{
    int legs;
}
```

এখন আর আমাদের অন্য দুইটি variable লিখার প্রয়োজন নেই। এখন অন্য একটি জাভা ফাইলে Dog class এর একটি অবজেক্ট তৈরি করে দেখি name আর age এর ব্যবহার করা যায় কিনা।

## Main.java

```
package package1;

public class Main {
    public static void main(String[] args) {
        Dog dog = new Dog();
        dog.name = "Moti";
    }
}
```

```

    dog.age = 5;
    dog.legs = 4;
}
}

```

ব্যবহার করা যাচ্ছে। এটাই ইনহেরিটেন্স। এভাবে মেথড কেও ইনহেরিট করা যায়।

### **Animal.java**

```

package package1;

public class Animal {
    String name;
    int age;
    void PrintInformation()
    {
        System.out.println("Name: "+name);
        System.out.println("Age: "+age);
    }
}

```

Animal ক্লাস এ একটি মেথড তৈরি করলাম PrintInformation নামে। এখন Dog class এর অবজেক্ট এর মাধ্যমে access করা যায় কিনা দেখি।

### **Main.java**

```

package package1;

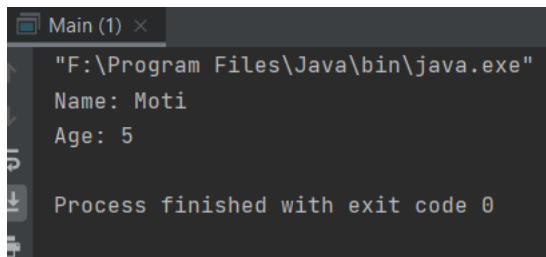
```

```

public class Main {
    public static void main(String[] args) {
        Dog dog = new Dog();
        dog.name = "Moti";
        dog.age = 5;
        dog.legs = 4;
        dog.PrintInformation();
    }
}

```

অ্যাক্সেস করা যাচ্ছে। আউটপুট হবে এমন।



The screenshot shows a console window titled "Main (1) x" with the following output:

```

"F:\Program Files\Java\bin\java.exe"
Name: Moti
Age: 5
Process finished with exit code 0

```

Leg এর মান দেখা যাচ্ছে না। Leg এর মান দেখতে হলে আমাদের method overriding শিখতে হবে।

একটি বিষয় জেনে রাখা ভাল। final class ইনহেরিট করা যায়না।

# অ্যাক্সেস মডিফায়ারস ও এনকেপ্সুলেশন

Modifier	Class	Package	Subclass	World
<b>public</b>	Yes	Yes	Yes	Yes
<b>protected</b>	Yes	Yes	Yes	No
<b>No modifier</b>	Yes	Yes	No	No
<b>private</b>	Yes	No	No	No

**public:** যেকোনো পাবলিক variable বা method কে আমরা প্রোজেক্ট এর যেকোনো জায়গা থেকে call করতে পারি।

**protected:** Protected variable বা method কে আমরা same class, same package থেকে কল করতে পারি। এবং different package যদি হয় তবে শুধু তার subclass থেকে access করতে পারি।

**No modifier:** No modifier variable বা method কে আমরা same class, same package থেকে কল করতে পারি। সে যদি ভিন্ন পেকেজ এ থাকে এবং সাব ক্লাস হয় তাহলেও তাকে অ্যাক্সেস করা যাবেনা।

তবে একই পেকেজ এর সাব ক্লাস হলে অ্যাক্সেস করা যাবে।

**private:** Private variable বা method কে আমরা শুধু same ক্লাস থেকে কল করতে পারি।

নিচের কোড গুলো দেখি

প্রথমে package1 এর আন্ডার এ কিছু ক্লাস খুলি

### **Access1.java**

```
package package1;
```

```
public class Access1 {
    public void print_ac1_public()
    {
        System.out.println("Print access 1 public");
    }

    protected void print_ac1_protected()
    {
        System.out.println("Print access 1 protected");
    }

    void print_ac1_nomodifier()
```

```

{
    System.out.println("Print access 1 no modifier");
}

private void print_ac1_private()
{
    System.out.println("Print access 1 private");
}
}

```

### **Access2.java**

```
package package1;
```

```

public class Access2 extends Access1{
    public static void main(String[] args) {
        Access1 access1 = new Access1();
        access1.print_ac1_public();
        access1.print_ac1_protected();
        access1.print_ac1_nomodifier();
        access1.print_ac1_private(); //error এটা হওয়ার ই ছিল কারন প্রাইভেট
        মেম্বার কে অন্য ক্লাস থেকে অ্যাক্সেস করা যায়না
    }
}

```

```

Access2 access2 = new Access2();
access2.print_ac1_public();
access2.print_ac1_protected();
access2.print_ac1_nomodifier();

```



```
access2.print_ac1_private(); //error
```

```
    }  
}
```

### **Access3.java**

```
package package1;
```

```
public class Access3 {  
    public static void main(String[] args) {  
        Access1 access1 = new Access1();  
        access1.print_ac1_public();  
        access1.print_ac1_protected();  
        access1.print_ac1_nomodifier();  
        access1.print_ac1_private(); //error । এটা হওয়ার ই ছিল কারন প্রাইভেট  
        মেম্বার কে অন্য ক্লাস থেকে অ্যাক্সেস করা যায়না
```

```
    }  
}
```

এখন package2 এর আন্ডার এ কিছু ক্লাস খুলি

### **Access4.java**

```
package package2;
```

```
import package1.Access1;
```

```

public class Access4 {
    public static void main(String[] args) {
        Access1 access1 = new Access1();
        access1.print_ac1_public();
        access1.print_ac1_protected(); //error যেহেতু পেকেজ এর বাইরে তাই
print_ac1_protected() কে অ্যাক্সেস করা যায়নি
        access1.print_ac1_nomodifier(); //error যেহেতু পেকেজ এর বাইরে তাই
print_ac1_nomodifier() কে অ্যাক্সেস করা যায়নি

        access1.print_ac1_private(); //error প্রাইভেট মেম্বার কে নিজ ক্লাস ছাড়া
অ্যাক্সেস করা যায়না
    }
}

```

### **Access5.java**

```

package package2;

import package1.Access1;

public class Access5 extends Access1 {
    public static void main(String[] args) {
        Access1 access1 = new Access1();

```

```
access1.print_ac1_public(); //পাবলিক কে যেকোনো জায়গা থেকে কল করা
```

যায়

```
access1.print_ac1_protected(); //error এটা চার্ট অনুযায়ী কাজ করার কথা
```

কিন্তু error দিচ্ছে। কারন হচ্ছে Access5 পেকেজ এর বাইরে আছে। যদিও সাব ক্লাস।

তবুও এক্ষেত্রে print\_ac1\_protected() এই মেথড কে অ্যাক্সেস করতে হলে যেই ক্লাস টি

Access1 কে Inherit করেছে সেই ক্লাসের অবজেক্ট দিয়ে print\_ac1\_protected()

মেথড কে কল করতে হবে। নিচে দেখানো হয়েছে

```
access1.print_ac1_nomodifier(); //error পেকেজ এর বাইরে তাই
```

```
access1.print_ac1_private(); //error ক্লাস এর বাইরে তাই
```

```
Access5 access5 = new Access5();
```

```
access5.print_ac1_protected(); //উপরে বলেছি পেকেজের বাইরে থাকলে সেই
```

ক্লাসের মেথড বা variable কে, যেই ক্লাস কে সাবক্লাস ডিক্লেয়ার করা হয়েছে ওই ক্লাসের

মাধ্যমেই access করা যাবে

```
access5.print_ac1_public(); //পাবলিক কে যেকোনো জায়গা থেকে কল করা
```

যায়

```
}
```

```
}
```

এতক্ষণ তো অবজেক্ট খুলে দেখলাম। এখন অবজেক্ট না খুলে দেখি inheritance এর

মাধ্যমে দেখি কোথায় কাকে inherit করা যায়।

**Access2.java** তে একটু পরিবর্তন করে দেখি

```
package package1;

public class Access2 extends Access1{
    void trytogetfrommom()
    {
        print_ac1_public();
        print_ac1_protected();
        print_ac1_nomodifier();
        print_ac1_private(); //error
    }
}
```

Access2 Access1 কে inherit করায় super class এর প্রাইভেট বাদে বাকি মেথড গুলো Access2 class এ inherit হয়ে যাচ্ছে।

এখন যদি **Access5.java** তে কিছু পরিবর্তন করি

```
package package2;

import package1.Access1;

public class Access5 extends Access1 {
    void trytogetfrommom()
    {
        print_ac1_public();
        print_ac1_protected();
        print_ac1_nomodifier(); //error
    }
}
```

```

        print_ac1_private(); //error
    }
}

```

এখানে `print_ac1_nomodifier()` কে সাব ক্লাসে থাকা সত্ত্বেও কল করা যায়নি কারন এটি `protected` এবং এটি ভিন্ন পেকেজ এ আছে।

আর প্রাইভেট কে অ্যাক্সেস করতে না পারার কারন ভিন্ন ক্লাস থেকে অ্যাক্সেস করার চেষ্টা করা হয়েছে।

এতক্ষন মেথড দিয়ে দেখলাম inheritance এর বেলায় access modifier এর ভূমিকা। এখন variable দিয়ে দেখি।

**Access1.java** তে কিছু variable নেই।

```

package package1;

public class Access1 {
    int nomod;
    protected int proc;
    public int publ;
    private int prvt;
}

```

**Access2.java** তে কিছু পরিবর্তন করি

```
package package1;

public class Access2 extends Access1{

    int x = proc;
    int y = publ;
    int z = nomod;
    int q = prvt; //error
}
```

Same package এর বেলায় private ছাড়া বাকি সব inherit করা যাচ্ছে।

এখন Access5.java তে কিছু পরিবর্তন করি।

```
package package2;

import package1.Access1;

public class Access5 extends Access1 {

    int x = proc;
    int y = publ;
    int z = nomod; //error
    int q = prvt; //error
}
```

ভিন্ন package এর বেলায় protected আর private বাদে বাকিদের inherit করা যাচ্ছে। যে মেথড গুলো inherit করতে পেরেছি সেই মেথড গুলো override ও করা যাবে।

এইসব inaccessible variable এবং method গুলো কে access করার জন্য আমরা getter এবং setter method ব্যবহার করি।

# গেটার এবং সেটার মেথড

নিচে একটি ক্লাস খুলে তাতে গেটার সেটার তৈরি করে দেখানো হল।

## **Animal.java**

```
public class Animal {  
    private String name;  
    private int leg;  
    private boolean tail;  
  
    //Constructor  
    public Animal(String namex, int legx, boolean tailx) {  
        name = namex;  
        leg = legx;  
        tail = tailx;  
    }  
  
    //getter for name  
    public String getName() {  
        return name;  
    }  
  
    //setter for name  
    public void setName(String namex) {
```



```
        name = namex;
    }

    //getter for leg
    public int getLeg() {
        return leg;
    }

    //setter for leg
    public void setLeg(int legx) {
        leg = legx;
    }

    //getter for tail
    public boolean isTail() {
        return tail;
    }

    //setter for tail
    public void setTail(boolean tailx) {
        tail = tailx;
    }
}
```

নিচের কোডটি name এর গেটার মেথড। এটি name এর value রিটার্ন করে দিচ্ছে।

```
public String getName() {
```

```

    return name;
}

```

নিচের কোডটি দিয়ে name এর value সেট করা যায়। এই মেথড টি parameter হিসেবে একটি স্ট্রিং নিয়ে name এ রেখে দেয়।

```

public void setName(String namex) {
    name = namex;
}

```

this keyword ব্যবহার করলে আরও সহজে করা যায়।

## **Animal.java**

```

public class Animal {
    private String name;
    private int leg;
    private boolean tail;

    //Constructor
    public Animal(String name, int leg, boolean tail) {
        this.name = name;
        this.leg = leg;
        this.tail = tail;
    }
}

```

```
//getter for name  
public String getName() {  
    return name;  
}
```

```
//setter for name  
public void setName(String name) {  
    this.name = name;  
}
```

```
//getter for leg  
public int getLeg() {  
    return leg;  
}
```

```
//setter for leg  
public void setLeg(int leg) {  
    this.leg = leg;  
}
```

```
//getter for tail  
public boolean isTail() {  
    return tail;  
}
```

```
//setter for tail  
public void setTail(boolean tail) {
```

```
    this.tail = tail;  
  }  
}
```

এখানে `this.name` দিয়ে বোঝানো হয় `Animal` class এর `name` কে। আর শুধু `name` সেটার মেথড এর `parameter` এর `name` কে নির্দেশ করে। `this` keyword নিয়ে পরবর্তীতে বিস্তারিত আলোচনা করা হবে।

# মেথড ওভাররাইডিং

ওভার রাইডিং এর মানে হচ্ছে একটি মেথডের বডি কে নতুন করে লিখা। অর্থাৎ তার ফাংশনালিটি পরিবর্তন করে দেওয়া। এটিকে রান টাইম পলিমরফিজম বলা হয়। কারণ এক্ষেত্রে ওভাররাইডেন ফাংশন এর কল রান টাইমে সংঘটিত হয়। রান টাইম মানে হচ্ছে প্রোগ্রামটি যখন রানিং বা চলমান থাকে। এর আরেক নাম Dynamic Method Dispatch.

## Animal.java

```
package com.niharon;
```

```
public class Animal {

    void PrintInfo()
    {
        System.out.println("Animal");
    }
}
```

শুরুতে Animal নামে একটি ক্লাস তৈরি করি। এর মধ্যে একটি মেথড তৈরি করি যার নাম PrintInfo।

## Main.java

```
package com.niharon;
```

```
public class Main {

    public static void main(String[] args) {

        Animal animal = new Animal();
        animal.PrintInfo();
    }
}
```

এখন Animal ক্লাসের অবজেক্ট তৈরি করে PrintInfo কে কল করলে Animal প্রিন্ট হবে।

## Dog.java

```
package com.niharon;

public class Dog extends Animal{
    @Override
    void PrintInfo() {
        System.out.println("Dog");
    }
}
```

এর পর একটি Dog ক্লাস তৈরি করলাম যা Animal ক্লাস কে ইনহেরিট করে। আর PrintInfo মেথড টি অভাররাইড করলাম। মেথড এর নাম আরগুমেন্ট এক থাকলে ইনহেরিট করলে ওই মেথডটি অভাররাইড হয়। @override এটি একটি ট্যাগ। এটি না দিলেও সমস্যা নেই। আমাদের যাতে বুঝতে সুবিধা হয় তাই দেওয়া হয়।

**Main.java**

```
package com.niharon;

public class Main {
    public static void main(String[] args) {
        Dog dog = new Dog();
        dog.PrintInfo();
    }
}
```

Dog ক্লাসের অবজেক্ট এর জন্য PrintInfo মেথডটি Dog প্রিন্ট করবে। কারণ Dog ক্লাসে PrintInfo মেথডকে অভাররাইড করে তার বডি পরিবর্তন করে দিয়েছি। অর্থাৎ

```
System.out.println("Animal");
```

এর জায়গায় `System.out.println("Dog");` এটি দিয়েছি।

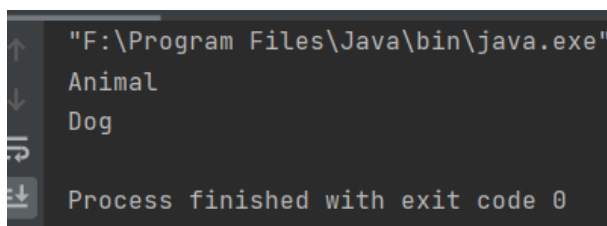
**Dog.java**

```
package com.niharon;

public class Dog extends Animal{
    @Override
    void PrintInfo() {
        super.PrintInfo();
        System.out.println("Dog");
    }
}
```

```
}  
}
```

আমরা যদি মাতৃ ক্লাসের `PrintInfo` কে কল করতে চাই তাহলে `super` keyword ব্যবহার করতে পারি। এটি মাতৃ ক্লাসের মেথড কে কল করবে। শুধু মেথড নয়, মাতৃ ক্লাসের যেকোনো `variable` কেও `super` keyword দিয়ে কল করা যায়। উপর কোড এর আউটপুট হবে এমন।



```
"F:\Program Files\Java\bin\java.exe"  
Animal  
Dog  
Process finished with exit code 0
```



# মেথড ওভারলোডিং

মেথড ওভাররাইডিং আর ওভারলোডিং কে অনেক এক করে ফেলি। দুইটি আসলে দুই জিনিস। মেথড ওভারলোডিং হচ্ছে এক নামের মেথড এর একাধিক রূপ। অর্থাৎ মেথড এর নাম যদি এক হয় এবং আর্গুমেন্ট ভিন্ন ভিন্ন হয় তাহলে তাকে মেথড অভারলোডিং বলা হয়। রিটার্ন টাইপ এক ও হতে পারে ভিন্ন ও হতে পারে। আর্গুমেন্ট গুলোর ডাটা টাইপ এক ও হতে পারে ভিন্ন ও হতে পারে। তবে এক হলে অবশ্যই সংখ্যায় ভিন্ন হতে হবে। আর্গুমেন্ট এর সংখ্যা ভিন্ন হতে পারে। এটিকে কম্পাইল টাইম পলিমরফিজম বলা হয়। কারণ এক্ষেত্রে ওভারলোডেড ফাংশন এর কল কম্পাইল টাইমে সংঘটিত হয়। কম্পাইল টাইম মানে হচ্ছে প্রোগ্রামটি যখন কম্পাইল হয়ে মেশিন কোডে রূপান্তরিত হয়।

## DoMath.java

```
package com.niharon;
```

```
public class DoMath {
```

```
    int sum (int x, int y) // এক
```

```
    {
        return x+y;
    }
```

```
    int sum (int x, int y, int z) // দুই
```

```
    {
```

```
    return x+y+z;
}
```

```
double sum (double x, double y) // তিন
{
    return x+y;
}
```

```
void sum() // চার
{
    System.out.println("No arguments");
}
}
```

উপরে sum মেথড এর সাহায্যে মেথড ওভারলোডিং এর উদাহরণ দেওয়া হল।

### **Main.java**

```
package com.niharon;

public class Main {

    public static void main(String[] args) {
        DoMath doMath = new DoMath();
        int x = doMath.sum(10,5);
        System.out.println(x);
    }
}
```

```
}
}
```

এটি এক নাম্বার sum মেথড কে কল করবে। যেহেতু এর দুইটি int আর্গুমেন্ট আছে।

## Main.java

```
package com.niharon;
```

```
public class Main {

    public static void main(String[] args) {
        DoMath doMath = new DoMath();
        int x = doMath.sum(10,5,6);
        System.out.println(x);
    }
}
```

এটি দুই নাম্বার sum মেথড কে কল করবে। যেহেতু এর তিনটি int আর্গুমেন্ট আছে।

## Main.java

```
package com.niharon;
```

```
public class Main {

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

```

    DoMath doMath = new DoMath();
    double x = doMath.sum(10.233,5.66);
    System.out.println(x);
}
}

```

এটি তিন নাম্বার sum মেথড কে কল করবে। যেহেতু এর দুইটি double আর্গুমেন্ট আছে।

## Main.java

```

package com.niharon;

public class Main {

    public static void main(String[] args) {
        DoMath doMath = new DoMath();
        doMath.sum();
    }
}

```

এটি চার নাম্বার sum মেথড কে কল করবে। যেহেতু এর কোন আর্গুমেন্ট নেই।

এই কাজগুলো Main ক্লাসেও করা যেত, তবে সব মেথড কে static ডিক্লেয়ার করতে হতো। কারন, main মেথড একটি static মেথড, এবং static মেথড থেকে non-static মেথড কে কল করা যায়না। static keyword নিয়ে পরবর্তীতে বিস্তারিত বলব। এবার দেখিয়ে দেই Main class এ কিভাবে করতাম কাজটি।

**Main.java**

```
package com.niharon;
```

```
public class Main {
```

```
    static int sum (int x, int y)
```

```
    {
```

```
        return x+y;
```

```
    }
```

```
    static int sum (int x, int y, int z)
```

```
    {
```

```
        return x+y+z;
```

```
    }
```

```
    static double sum (double x, double y)
```

```
    {
```

```
        return x+y;
```

```
    }
```

```
    static void sum()
```

```
    {
```

```
        System.out.println("No arguments");
```

```
    }
```

```
public static void main(String[] args) {  
    int x = sum(5,6);  
    int y = sum(5,6,7);  
    double z = sum(5.62,6.12);  
    System.out.println(x+" "+y+" "+z);  
    sum();  
}  
}
```

# কমান্ড লাইন আর্গুমেন্ট