

মাতৃভাষায় জাভা শিখি

আব্দুজ জামি



মাতৃভাষায় জাভা শিখি

আব্দুজ জামি

পিডিএফ সংস্করণ - ২.০

মাতৃভাষায় জাভা শিখি
গ্রন্থ সস্ব : আব্দুজ জামি
প্রচ্ছদ: [পারভেজ আহমেদ](#)

প্রথম প্রকাশ : অক্টোবর ২০২১
দ্বিতীয় প্রকাশ : নভেম্বর ২০২২

মূল্য :

পিডিএফ সংস্করণ : নির্দিষ্ট মূল্য নেই তবে চাইলে এক কাপ চা ☕ খাওয়াতেই পারেন

বিকাশ: 01521704287

রকেট: 01706335199

নগদ: 01521704287

বইটির পিডিএফ ভাৰ্সন ডাউনলোড করতে নিচের লিংকে যান অথবা QR code টি স্ক্যান করুন।

<https://niharon-pub.netlify.app/>



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 ইন্টারপ্রেটার 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 আব্দুজ জামি। 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। স্কুলও 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। বইটির পিডিএফ এর শুভেচ্ছা মূল্য মাত্র ২০ টাকা। তবে কারো মূল্য প্রদানে সমস্যা থাকলে সে নির্বিশেষে বিনামূল্যে বইটি পরতে পারে। বইটি সম্পর্কে আপনার মতামত জানাতে আমাকে মেইল করতে পারেন। আপনাদের সাড়া পেলে খুব শিগ্রই বইটির হার্ড কপি আপনাদের সামনে নিয়ে আসব। বইটির পিডিএফ এই লিঙ্কে পাওয়া যাবে

<https://niharon-pub.netlify.app/>

সুচিপত্র

জাভা কি কেন শিখব
 অবজেক্ট ওরিয়েন্টেড প্রোগ্রামিং
 জাভা IDE
 JDK
 জাভা ফাইলের গঠন
 IDE তে জাভা শুরু করা
 প্রিন্ট করা
 স্কোপ সিকুয়েন্স
 কমেন্টস
 ডাটা টাইপস
 ইউজার থেকে ইনপুট নেওয়া
 ফরমেট স্পেসিফায়ার
 অপারেটর
 কন্ডিশনাল লজিক
 লুপ
 অ্যারে
 মেথড
 Math ক্লাসের কিছু মেথড
 স্ট্রিং
 জাভা প্রজেক্ট এর গঠন
 প্যাকেজ
 ক্লাস এবং অবজেক্ট
 রেফারেন্স টাইপের মেমোরি বন্টন
 কন্সট্রাক্টর
 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

this কী-ওয়ার্ড

super কী-ওয়ার্ড

static ভেরিয়েবল

static মেথড

static ক্লাস

final ভেরিয়েবল

final মেথড

final ক্লাস

Static Final Variable

StringBuffer

StringBuilder

ArrayList

ArrayList sorting

HashMap

HashSet

PriorityQueue

Anonymous class

Abstract class

Interface

Lambda expression

Exception handling

extends vs implements

ফাইল

Multi-threading

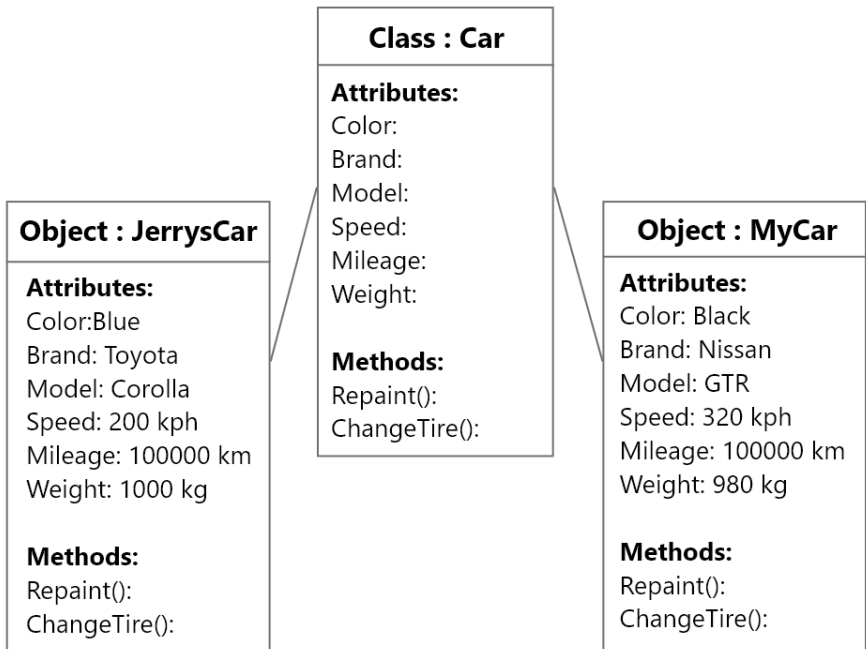
জাভা কি কেন শিখব

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। বস্তুভিত্তিক 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 কিছুই 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 গুলোও 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 বিন্ড 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 বিন্ড 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 রক্ষণাবেক্ষণে 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। এনক্যাপ্সুলেশন (Encapsulation)
- ৩। পলিমরফিজম (Polymorphism)

৪। অ্যাবস্ট্রাকশন (Abstraction)

জাভায় এই চারটি মূলনীতির সবগুলোই রয়েছে। তাই জাভাকে সম্পূর্ণ অবজেক্ট ওরিয়েন্টেড প্রোগ্রামিং ল্যাঙ্গুয়েজ বলা হয়।

জাভা IDE

IDE এর পূর্ণরূপ হল Integrated Development Environment। এর মধ্যে আমরা সাধারণত কোড লিখি। আই ডি ই হল এমন একটি সফটওয়্যার যা সব ধরনের ডেভেলপমেন্ট এর টুলস গুলোকে একত্রে করে আমাদের প্রোগ্রামিং বা ডেভেলপমেন্ট কে সহজ করে। জাভার জনপ্রিয় বেশ কয়েকটি আই ডি ই আছে যেমন-

১। IntelliJ IDEA

২। NetBeans

৩। Microsoft Visual Studio Code

৪। Eclipse ইত্যাদি।

যার যেটায় ভাল লাগে সেটা ব্যবহার করলেই হবে। তবে আমার নিজের IntelliJ IDEA টা বেশি ভাল লাগে। তার পর সবচেয়ে ভাল লাগে NetBeans। NetBeans একটি lightweight আই ডি ই। আর IntelliJ একটি শক্তিশালী আই ডি ই। নিচে এদের সবার অফিসিয়াল ডাউনলোড লিঙ্ক দেওয়া হল।

IntelliJ IDEA

<https://www.jetbrains.com/idea/>

NetBeans

<https://netbeans.apache.org/download/>

Visual Studio Code

<https://code.visualstudio.com/>

Eclipse

<https://www.eclipse.org/downloads/>

JDK

JDK হল জাভা প্রোগ্রামিং ভাষা ব্যবহার করে অ্যাপ্লিকেশন, অ্যাপলেট এবং কম্পোনেন্ট তৈরির জন্য একটি পরিবেশ। JDK তে জাভা প্রোগ্রামিং ভাষায় লেখা এবং জাভা প্ল্যাটফর্মে চলার প্রোগ্রামগুলির ডেভেলপমেন্ট ও পরীক্ষার জন্য দরকারী টুলসগুলো থাকে। জাভায় কোড করার পূর্বশর্ত হল JDK ইন্সটল করা। JDK ডাউনলোড এর লিঙ্ক নিচে দেওয়া হল।

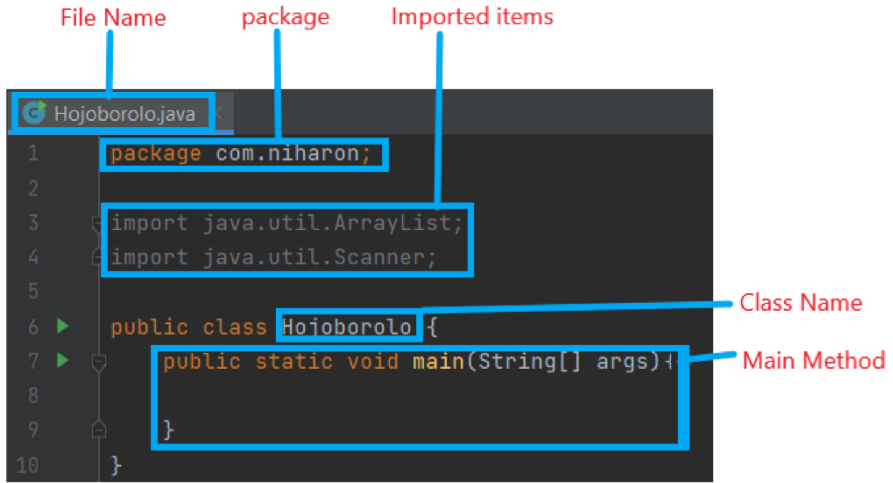
<https://www.oracle.com/java/>

এই বইটি লেখার সময়ে JDK এর latest ভার্সনটি ১৭। আপডেটেড ভার্সনটি নামানোর পরামর্শ দেওয়া হল।
নিচের QR code টি স্ক্যান করলে অথবা লিঙ্কে গেলে JDK ইন্সটল করার একটি টিউটোরিয়াল পাওয়া যাবে।



https://youtu.be/EPCEXgyP_OI

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। একটি কথা বলে রাখি। এই বইয়ে আমি কোড লিখার সময় সবার উপরে বোল্ড অক্ষরে যেটি লিখেছি তা হচ্ছে ফাইল নেম। এটি কোডের কোনো অংশ নয়। যেমন-

Main.java

```

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

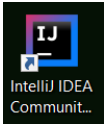
```
}
```

এখানে **Main.java** এটি আসলে ফাইলের নাম। এই ফাইলের প্রধান ক্লাসটির নাম ও ফাইলের নামের অনুরূপ হতে হবে।

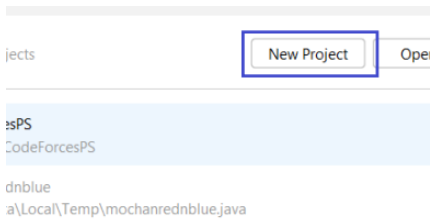
IDE তে জাভা শুরু করা

আমি IDE হিসেবে JetBrains এর IntelliJ ব্যবহার করেছি এই বই এর কোড লিখার সময়। তো আমি এটা দিয়েই দেখাচ্ছি প্রোজেক্ট খোলা। যে কেও যেকোনো IDE ব্যবহার করতে পারে।

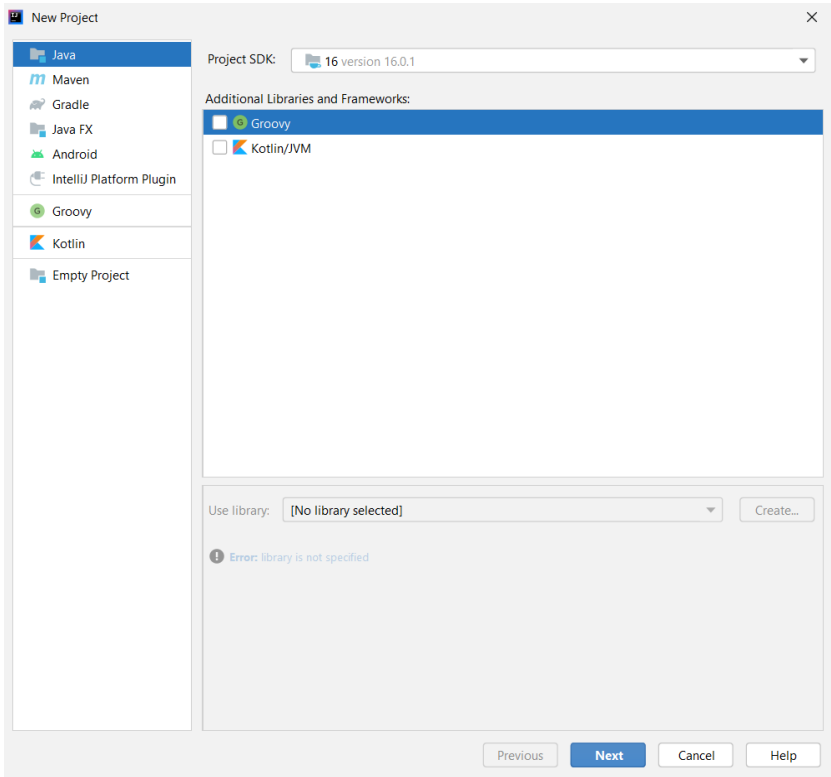
শুরুতেই IntelliJ Idea ওপেন করি।



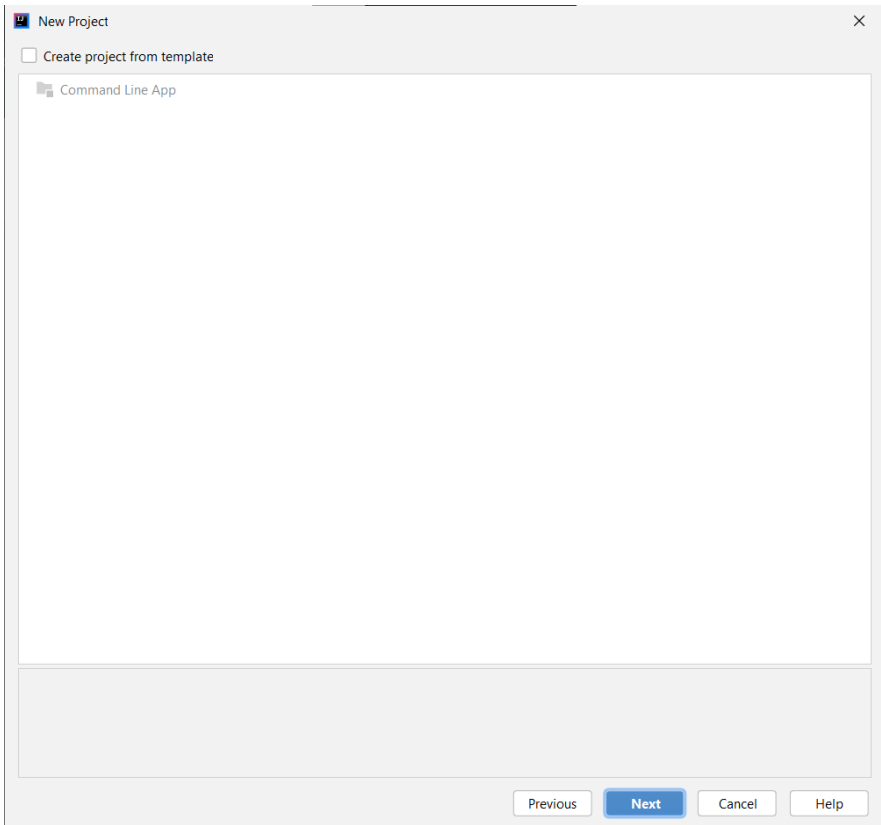
এর পর একটি Pannel আসবে। সেখানে New Project এ ক্লিক করি।



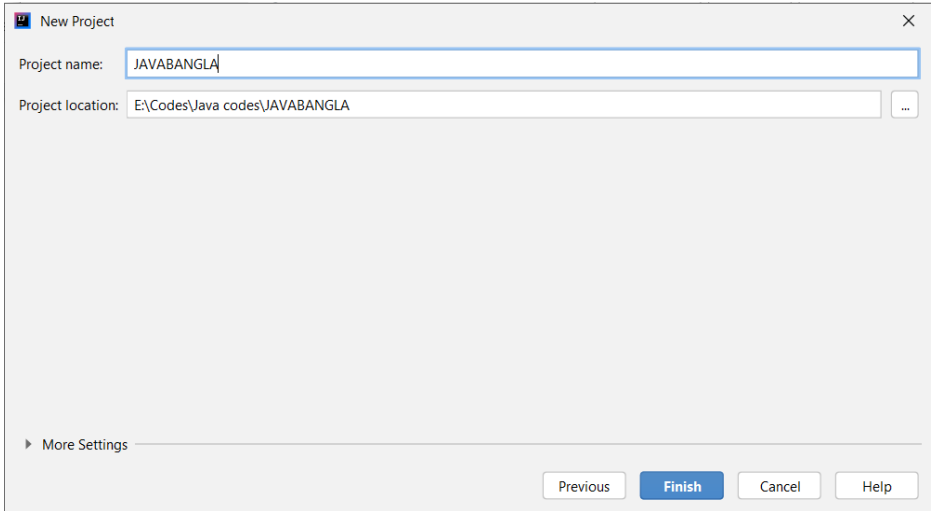
আবার ও একটি প্যানেল আসবে। সেখানেও Next এ ক্লিক করি।



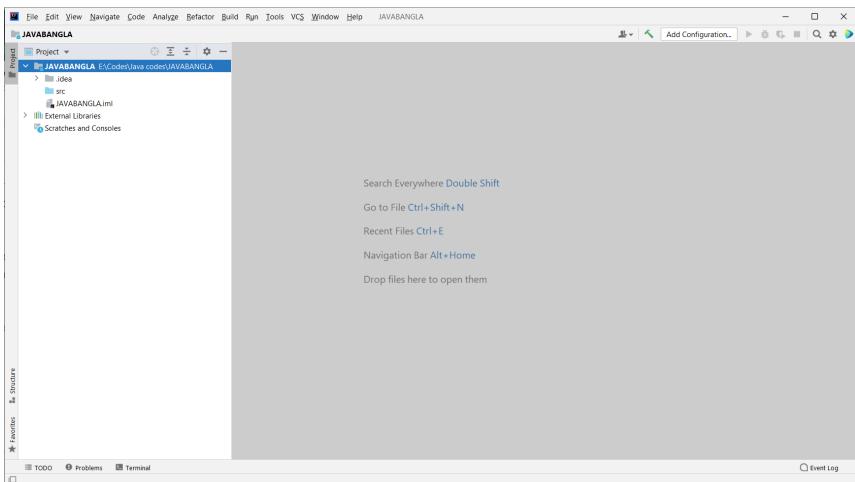
এর পরেও আরেকটি প্যানেল আসবে সেখানেও Next বাটন এ ক্লিক করি।



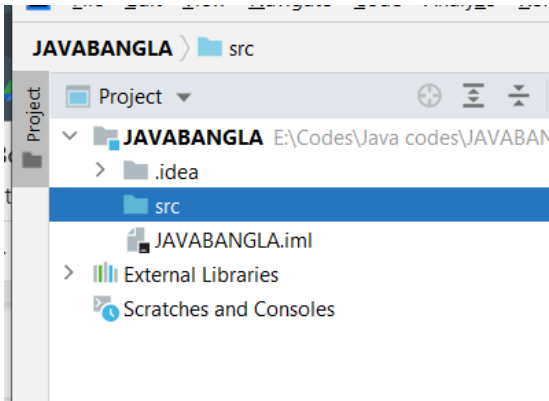
সব শেষে একটি উইন্ডো আসবে যেখানে ফাইলের একটি নাম এবং ফাইল লোকেশন দিয়ে Finish বাটনে ক্লিক করলেই নতুন প্রোজেক্ট তৈরি হয়ে যাবে।



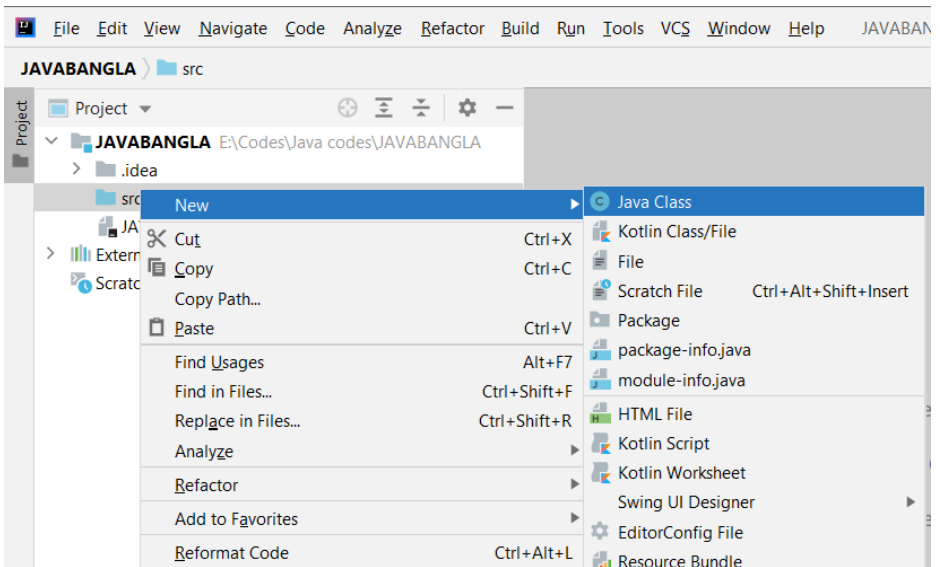
এরপর নিচের মত একটি উইন্ডো আসবে।



এখন বামে উপরে প্রোজেক্ট নেইম এর নামে যেই ফোল্ডারটি দেখা যায় তাকে expand করি। Expand করার জন্য > এই চিহ্নে ক্লিক করতে হবে।

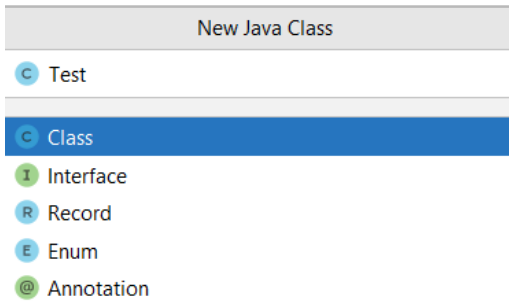


এরপর src নামের ফোল্ডার এ mouse এর right বাটনে ক্লিক করলে যে পপ আপ আসবে তাতে new এ কার্সর রাখলে আরও একটি পপ আপ আসবে।

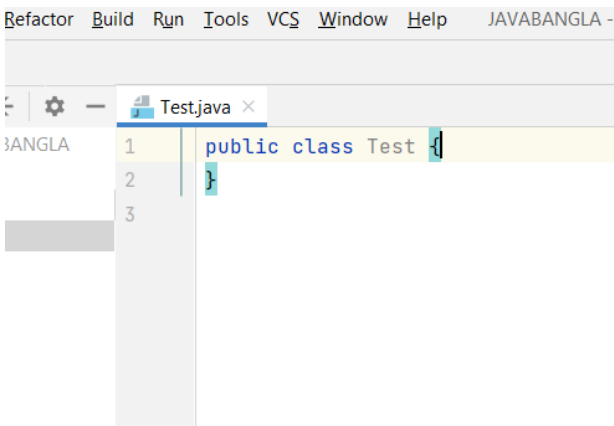


সেখানে java class এ ক্লিক করলে আরেকটি পপ আপ আসবে।

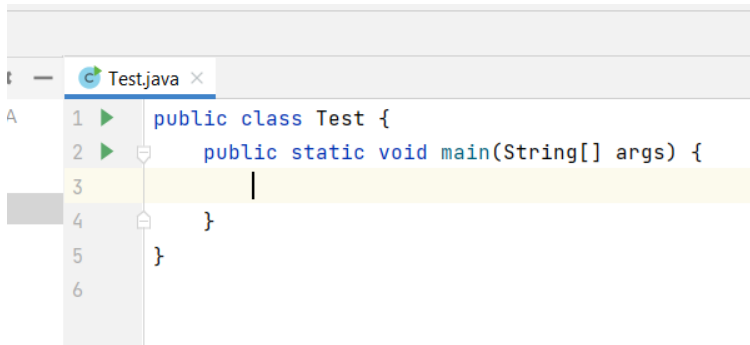
ওই পপ আপ এর class অপশন এ সিলেক্টেড থাকা অবস্থায় java class এর একটি নাম দিয়ে Enter দিলেই একটি নতুন জাভা ক্লাস খুলে যাবে।



এটি হচ্ছে একটি জাভা ক্লাস। এখানেই আমরা যাবতীয় কোড লিখবো।

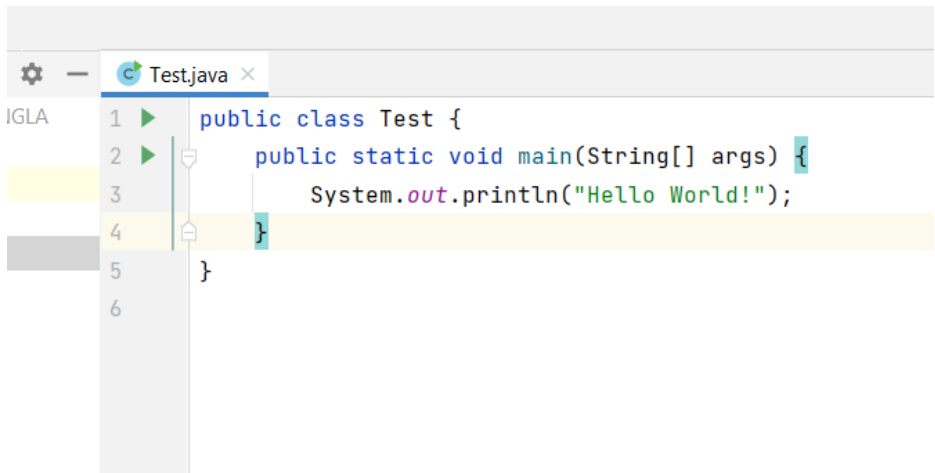


জাভা ক্লাসের ভিতরে একটি মেইন মেথড লিখে হবে। `public static void main(String[] args){}` এটি ই হচ্ছে মেইন মেথড। এটি লেখার জন্য IntelliJ Idea এ একটি শর্ট কাট আছে। এটি হল `psvm` লিখে `tab / enter` বাটনে প্রেস করলেই মেইন মেথড তৈরি হয়ে যাবে।



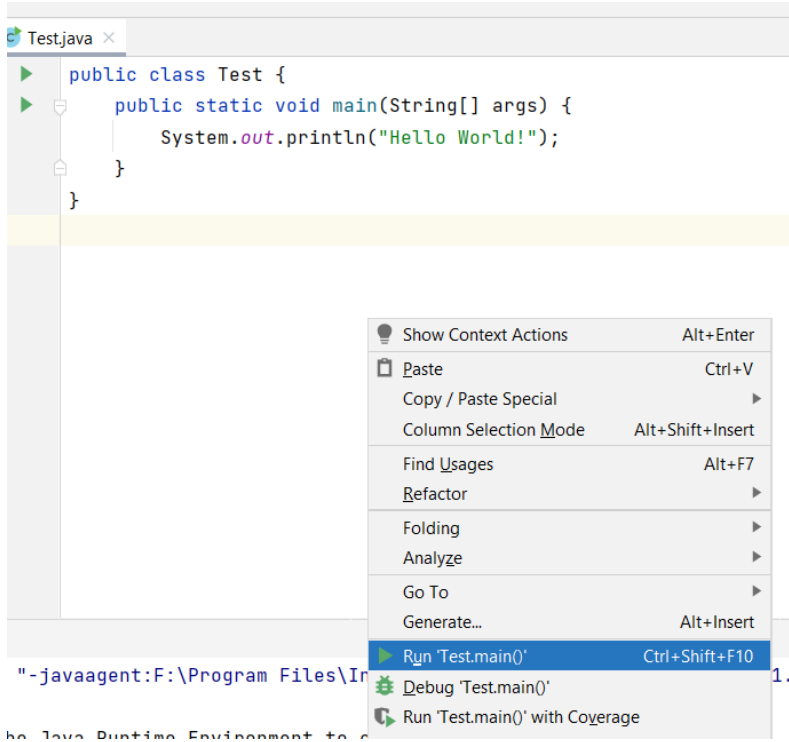
```
Test.java x
1 public class Test {
2     public static void main(String[] args) {
3
4     }
5 }
6
```

মেইন মেথডের ভিতরে বাইরে সব খানেই কোড লিখতে হবে। আমরা বেসিক কোডগুলো মেইন মেথডেই লিখবো।

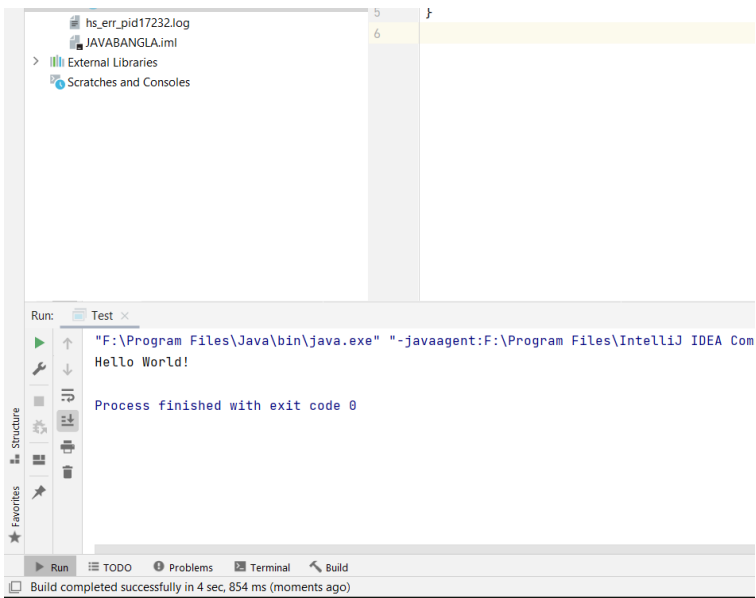


```
Test.java x
1 public class Test {
2     public static void main(String[] args) {
3         System.out.println("Hello World!");
4     }
5 }
6
```

কোড লেখার পর আউটপুট দেখার জন্য ওই ফাইলেই কোড লেখার জন্য যে জায়গা আছে সেখানেই মাউসের রাইট বাটনে ক্লিক করলে একটি পপ আপ আসবে। সেখানে Run File Name নামে একটি অপশন আছে। সেখানে ক্লিক করলে জাভা ফাইলটি রান হবে।



আউটপুট দেখা যাবে নিচের অংশে।



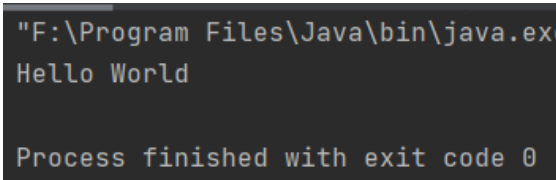
প্রিন্ট করা

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
```

অনুশীলনী

১। নিচের লিখাটি প্রিন্ট করার চেষ্টা করি

Matru Vashay Java Shikhi

২। নিজের সম্পর্কে তথ্যগুলো লিখার চেষ্টা করি যেমনঃ

Name: Abduz Zami

Age: 21

[বিঃদ্রঃ দ্বিতীয় লাইনের আগের লাইন ব্রেক এর জন্য আমরা '\n' অথবা println() ব্যবহার করতে পারি।]

স্কেপ সিকুয়েন্স

কতগুলো বিশেষ character আছে যার সামনে '\' থাকে এবং যা কম্পাইলার এর কাছে কিছু বিশেষ অর্থ বহন করে। এদেরকেই বলে স্কেপ সিকুয়েন্স। নিচে জাভার স্কেপ সিকুয়েন্সগুলো দেখানো হল।

স্কেপ সিকুয়েন্স	বর্ণনা
\t	একটি ট্যাব বা চারটি স্পেস তৈরি করে
\b	এটি একটি ব্যাকস্পেস তৈরি করে
\n	এটি একটি নতুন লাইন তৈরি করে
\r	এটি কার্সরকে লাইনের শুরুতে নিয়ে যায়
\f	এটি একটি ফর্ম ফিড ইনসার্ট করে
\'	একটি (') character ইনসার্ট করে
\"	একটি (") character ইনসার্ট করে
\\	একটি backslash(\) ইনসার্ট করে
\u	Unicode বসাতে ব্যবহার করা হয়

নিচে কয়েকটি উদাহরণ দেখানো হল।

Test.java

```
public class Test {
    public static void main(String[] args) {
        System.out.println("Matrivasahy\nJava Shikhi\nBy Abdur Zami");

        System.out.println("Matrivasahy\tJava Shikhi\tBy Abdur Zami");
    }
}
```

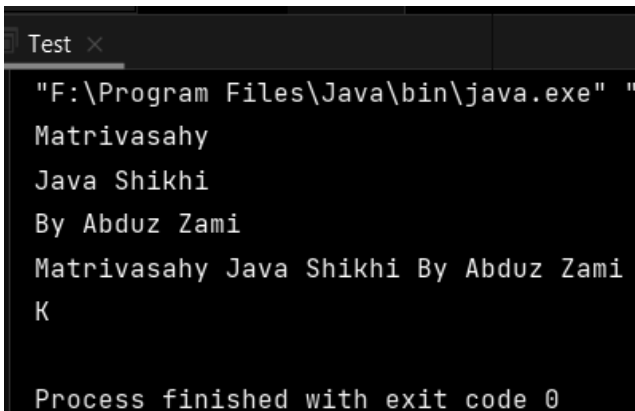
```

char ch = '\u039A';
System.out.println(ch);
}
}

```

এখানে 039A হল K এর ইউনিকোড।

আউটপুট:



```

Test x
"F:\Program Files\Java\bin\java.exe" "
Matrivasahy
Java Shikhi
By Abduz Zami
Matrivasahy Java Shikhi By Abduz Zami
K
Process finished with exit code 0

```

কতগুলো character আছে যা কিছু বিশেষ নির্দেশ বহন করে। যেমন: ‘ “ \ () ইত্যাদি। এদেরকে যদি প্রিন্ট করতে হয় তাহলে এদের আগে একটি \ backslash দিতে হয়। যেমন যদি একটি ফাইল পথ প্রিন্ট করতে চাই তাহলে নিচের মত করে লিখতে হবে।

```
System.out.println("F:\\Program Files\\Java\\bin");
```

এর আউটপুট হবে নিচের মত।

```

"F:\Program Files\Java\bin\j
F:\Program Files\Java\bin

```

কমেন্টস

কমেন্ট হচ্ছে কোডের এমন কিছু লাইন যা কম্পাইলার রান করেনা। অর্থাৎ ignore করে। কমেন্টে আমরা অনেক hints লিখে রাখতে পারি যা পরে কোডটি বুঝতে সহায়তা করবে। কমেন্ট করার দুইটি উপায় আছে জাভায়।

একটি হল //

এবং অন্যটি হল /* */

// দিয়ে কমেন্ট লিখলে ওই লাইনটি আর execute হবেনা।

আর /* */ দিয়ে কমেন্ট করলে /* এবং */ এর ভিতরের লেখাগুলো execute হবেনা।

এটা কোডের মাধ্যমে দেখালে আরও স্পষ্ট হবে।

```
public class Test {
    public static void main(String[] args) {
        System.out.println("START");

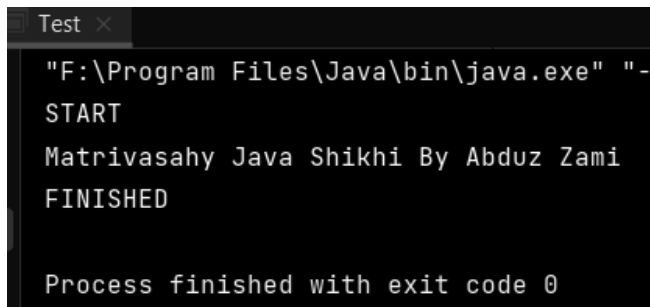
        //System.out.println("Matrivasahy\nJava Shikhi\nBy Abdur Zami");

        System.out.println("Matrivasahy\tJava Shikhi\tBy Abdur Zami");

        /*
        char ch = '\u039A';
        System.out.println(ch);
        */

        System.out.println("FINISHED");
    }
}
```

এর আউটপুট হবে এমন।



```
Test x
"F:\Program Files\Java\bin\java.exe" "-
START
Matrivasahy Java Shikhi By Abduz Zami
FINISHED

Process finished with exit code 0
```

যেগুলো কমেন্ট করা হয়েছে সেগুলো কিন্তু execute হয়নি।

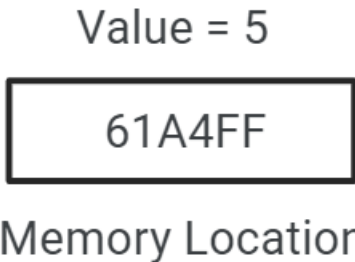
ডাটা টাইপস

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।

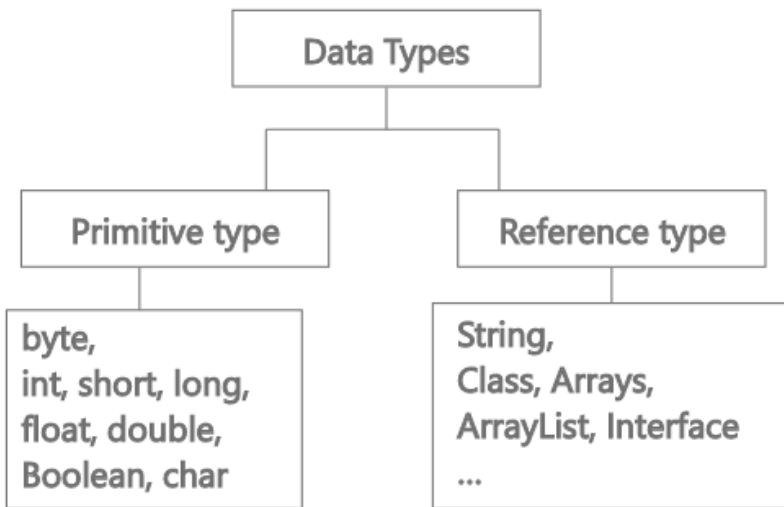
জাভায় মূলত দুই ধরনের ডাটা টাইপ আছে। একটি হচ্ছে প্রিমিটিভ টাইপ, অন্যটি নন প্রিমিটিভ বা রেফারেন্স টাইপ। আমরা যারা C, C++ ল্যঙ্গুয়েজ এর সাথে পরিচিত, তারা যে ডাটা টাইপ গুলো ব্যবহার করে এসেছি (int, float, double, long long int, unsigned long long) এসব ই ছিল প্রিমিটিভ টাইপ।

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 = 1000000000;
```

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 = 1000000000000000000L;
```

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;
```

ভগ্নাংশগুলো 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 পারবেনা। কম্পাইলার ধরে নিবে এটি একটি double. কারন 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। character কে single quotation (' ') এর ভিতরে রাখতে হয়। 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 টেক্সট 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);
```

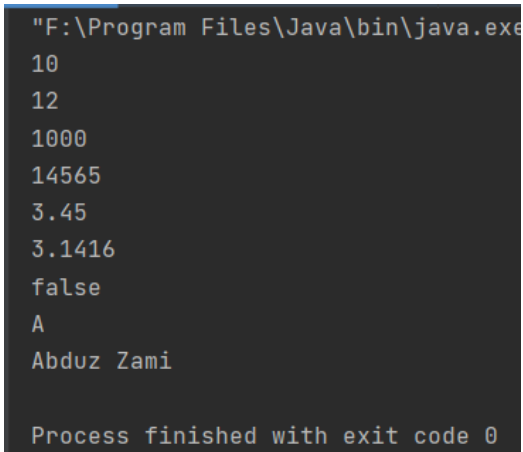
```

        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। কিন্তু

int টাইপের variable। এখানে int কে String এ casting করে ফেলেছে। এমনকি শুধু i প্রিন্ট করলেও implicit type casting হয়ে যাচ্ছে।

টাইপ কাস্টিং

টাইপ কাস্টিং (type casting) একটি গুরুত্বপূর্ণ বিষয় যেকোনো প্রোগ্রামিং ভাষাতেই। টাইপ কাস্টিং মানে হল এক ডাটা টাইপের variable কে অন্য টাইপে কাস্ট করা বা পরিবর্তন করা। তবে এক্ষেত্রে শর্ত হল ওই variable এর সাইজ যেই টাইপে নিতে হবে তার থেকে ছোট বা সমান হতে হবে।

Test.java

```
public class Test {
    public static void main(String[] args) {
        double x = 3.456;
        int y = (int) x;
    }
}
```

এখানে double টাইপের variable কে int এ কাস্ট করা হয়েছে।

অনুশীলনী

১। উপরে বর্ণিত ডাটা টাইপ গুলো নিজে নিজে ডিক্লেয়ার করে প্রিন্ট করার চেষ্টা করি।

২। একটি int কে অন্য একটি int দিয়ে ভাগ করে ভাগফলকে float অথবা double টাইপের variable এ রাখি যেন precision বা দশমিকের পরের সংখ্যা গুলো ঠিকমত দেখায়।

[বিঃদ্রঃ এখানে টাইপ কাস্টিং ব্যবহার করতে হবে।]

এই বিষয়টি অনেকের কাছে নতুন হওয়ায় আমি এখানেই উত্তর বলে দিচ্ছি। একটি int কে int দিয়ে ভাগ করলে টা একটি int ই রিটার্ন করে। তাই আমাদের হরের বা লবের যেকোনো একটিকে double বা float এ টাইপ কাস্টিং করতে হবে। নিচে কোড করে দেখাচ্ছি।

Test.java

```
public class Test {  
    public static void main(String[] args) {  
        double x,u;  
        int y = 5,z = 12;  
        x = z / (double) y;  
        u = (double) y/z;  
        System.out.println(x);  
        System.out.println(u);  
    }  
}
```


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 টারমিনাল 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

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। কারন 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

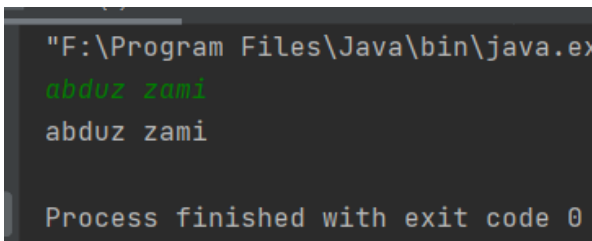
```
import java.util.Scanner;

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

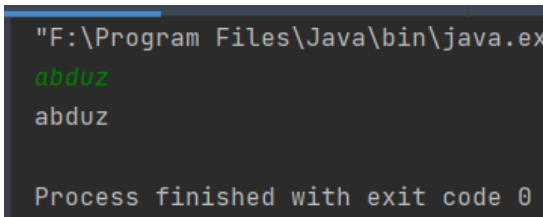
```
import java.util.Scanner;

public class Main{

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

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

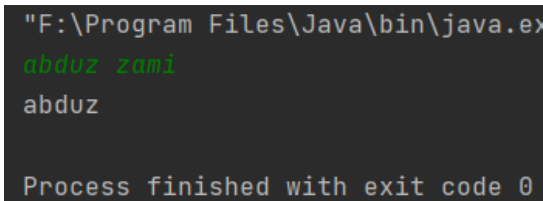
এর output:



```
"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
```

কারন next() space পেলে ব্রেক হয়।

Scanner.nextLine() ব্যবহারের সতর্কতা

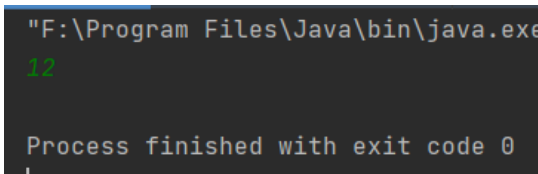
Main.java

```
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() গ্রহন 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।

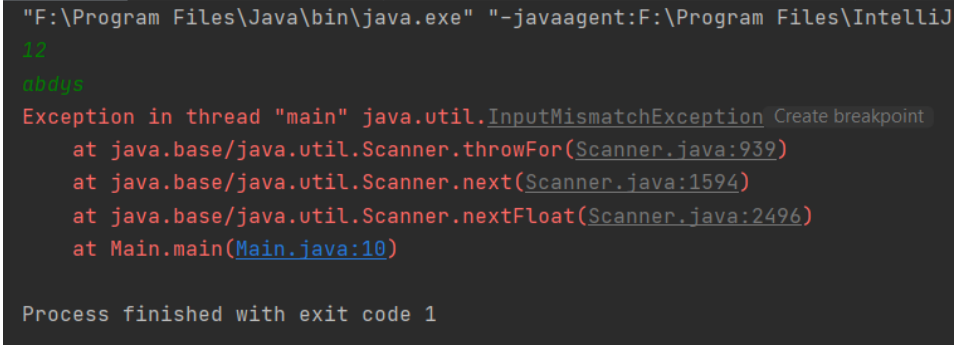
Main.java

```
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 গৃহীত 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;

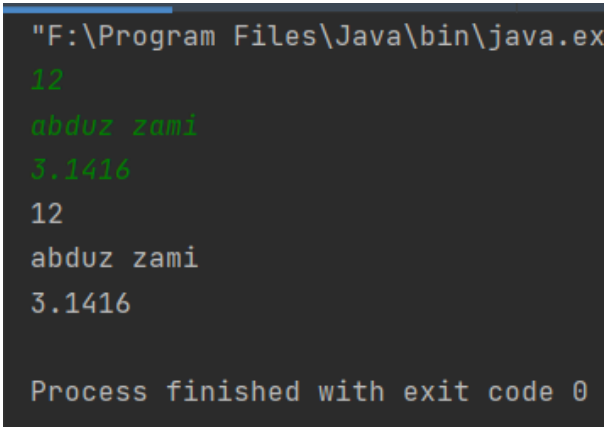
public class Main{

    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।

অনুশীলনী

১। নিচের প্রতিটি ডাটা টাইপের variable ইউজার ইনপুট নিতে হবে।

byte	int	short	long	double	float	boolean	char	String
------	-----	-------	------	--------	-------	---------	------	--------

২। নিচের variable গুলো ক্রমানুযায়ী ইনপুট নিতে হবে

```
Int x;  
String s;  
double d;  
String z;  
byte b;  
char c;
```

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 সাইন্টফিক যেকোনো একটি পদ্ধতি তে প্রিন্ট করে। সংখ্যাটি যদি 10^6 এর থেকে ছোট হয়ে তাহলে দশমিক পদ্ধতিতে প্রিন্ট করে। অন্যথায় সাইন্টফিক 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।



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
```

অনুশীলনী

১। ফরম্যাট স্পেসিফায়ারের সাহায্যে ইচ্ছামত ডাটা টাইপের variable নিয়ে টা স্ক্যান করার পর প্রিন্ট করতে হবে।

২। নিচের মত একটি টেবিল প্রিন্ট করতে হবে।

January	February	March	April
May	June	July	August
September	October	November	December

অপারেটর

জানার 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 সমান বাড়ায়	x+=5	x=x+5
-=	= 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	x/=5	x=x/5

	পাশের মান দ্বারা ভাগ করে বাম পাশের চলকে রাখা হয়		
%=	= এর বাম পাশের চলকের মান ডান পাশের মান দ্বারা ভাগ করলে যে ভাগশেষ হয় তাকে বাম পাশের চলকে রাখা হয়	$x\%=5$	$x=x+5$
&=	= 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	$x =5$	$x=x 5$
^=	= 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 অপারেশন করে	$x>>=2$	$x=x>>2$

	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	$x > y$

		বড় হলে true রিটার্ন করে	
<	Less than	বামপক্ষ ডানপক্ষ থেকে ছোট হলে true রিটার্ন করে	$x < y$
>=	Greater than or equal to	বামপক্ষ ডানপক্ষ থেকে বড় বা সমান হলে true রিটার্ন করে	$x \geq y$
<=	Less than or equal to	বামপক্ষ ডানপক্ষ থেকে ছোট বা সমান হলে true রিটার্ন করে	$x \leq y$

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 AND অপারেশন করে	$5 = 101$ $3 = 011$ $5 \& 3 = 001 = 1$
	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 বিটগুলোকে বাম দিকে শিফট করে।	$5 = 101$ $5 << 2 = 10100$ $10100 = 20$
>>	Right shift	একটি সংখ্যার বিটগুলোকে ডান দিকে শিফট করে।	$5 = 101$ $5 >> 2 = 001 = 1$ $-5 =$ 1111111111111111 111111111111011

			$-5 \gg 2 =$ 111111111111111111 1111111111111110 $= -2$ 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।
>>>	Unsigned Right Shift	একটি সংখ্যার বিট গুলোকে ডান দিকে শিফট করে। সাইন বিট সহ	$-5 =$ 111111111111111111 111111111111011 $-5 \gg 2 =$ 001111111111111111 1111111111111110 $= 1073741822$ Unsigned right shift করলে সাইন বিট ডিও শিফট হয়।

অনুশীলনী

- ১। দুইটি সংখ্যার মধ্যে যোগ, বিয়োগ, গুণ ও ভাগ করার একটি প্রোগ্রাম লিখতে হবে।
- ২। দুইটি বা ততোধিক সংখ্যার মধ্যে সবচেয়ে বড় এবং সবচেয়ে ছোট সংখ্যাটি খুঁজে নিয়ে আউটপুট দেখাতে হবে।
- ৩। একটি সংখ্যা ৩ এবং ৫ দ্বারা বিভাজ্য কিনা যাচাই করতে হবে।

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।

if-else

If-else দিয়ে আমরা কোন শর্তে আমাদের প্রোগ্রাম কোন কাজ করবে তা ঠিক করে দিতে পারি। নিচে কতগুলো উদাহরণ দিয়ে আলোচনা করছি।

booktest.java

```
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 উদাহরণ দেখি।

booktest.java

```
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।

booktest.java

```

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।

booktest.java

```
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।

booktest.java

```
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।

booktest.java

```

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।

booktest.java

```

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।

সুইচ

সুইচ if-else এর একটি বিকল্প হতে পারে। তবে সবক্ষেত্রে নয়। if-else আমাদের অনেক বেশি flexibility দেয়। নিচে সুইচের কিছু উদাহরণ দিয়ে আলোচনা করা হল।

javabook.java

```
public class javabook {
    public static void main(String[] args) {
        int n = 5;
        switch (n){
            case 1:
                System.out.println("This is 1");
                break;
            case 5:
                System.out.println("This is a 2");
                break;
            default:
                System.out.println("Invalid");
        }
    }
}
```

উপরের কোডে n যদি 1 হয় তবে This is 1 প্রিন্ট হবে। মনে রাখতে হবে সুইচে প্রতিটি কেইস এর শেষে ব্রেক দিতে হবে। default এর বেলায় লাগবেনা। কারণ এর পর আর কোন কেস থাকবেনা। ব্রেক না দিলে কি সমস্যা হয় সেটা দেখি।

javabook.java

```
public class javabook {
    public static void main(String[] args) {
        int n = 1;
        switch (n){
            case 1:
```

```

        System.out.println("This is 1");
    case 2:
        System.out.println("This is 2");
    case 5:
        System.out.println("This is 3");
        break;
    default:
        System.out.println("Invalid");
    }
}
}

```

Output:

This is 1

This is 2

This is 3

এরকম টা কেন হল? এরকম টা হয়ার কারণ হচ্ছে ব্রেক না দেওয়া। বেক না দেওয়ার জন্য ওই কেস থেকে শুরু করে অন্য কোন কেস এর ব্রেক বা একদম শেষ পর্যন্ত চলে যাচ্ছে। তাই আমাদের সুইচের বেলায় ব্রেক এর বেপারে সতর্ক থাকতে হবে।

আরেকটি গুরুত্বপূর্ণ কথা। সুইচে float বা double দিয়ে কেস তৈরি করা যায়না।

এবার character দিয়ে দেখি।

javabook.java

```

public class javabook {
    public static void main(String[] args) {
        char ch = 'a';
        switch (ch){
            case 'a':
                System.out.println("This is an A");
                break;

```

```

        case 'b':
            System.out.println("This is a B");
            break;
        default:
            System.out.println("Invalid");
    }
}
}

```

আমরা চাইলে কতগুলো কেস একসাথে লিখতে পারি যদি তাদের কাজ একরকম হয়।

javabook.java

```

public class javabook {
    public static void main(String[] args) {
        char ch = 'f';
        switch (ch){
            case 'a':
            case 'f':
            case 'g':
                System.out.println("This is an A");
                break;
            case 'b':
                System.out.println("This is a B");
                break;
            default:
                System.out.println("Invalid");
        }
    }
}

```

এখানে a,f,g কেস এর কাজ একই। তাই এদেরকে একসাথে লেখা হয়েছে।

Enhanced switch

Enhanced switch এর সাহায্যে কোডটিকে ছোট করে ফেলা যায়। নিচে একটি উদাহরণ দেওয়া হল।

javabook.java

```
public class javabook {
    public static void main(String[] args) {
        char ch = 'a';
        switch (ch) {
            case 'a', 'f', 'g' -> System.out.println("This is an A");
            case 'b' -> System.out.println("This is a B");
            default -> System.out.println("Invalid");
        }
    }
}
```

অনুশীলনী

- ১। একটি character vowel কিনা যাচাই করতে হবে।
- ২। একটি সংখ্যা ঋণাত্মক নাকি ধনাত্মক যাচাই করতে হবে।
- ৩। একটি সংখ্যা মৌলিক কিনা যাচাই কতে হবে।

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।

জানায় 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 ও 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 দিগুণ হারে বারাবারে 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 কমে কমে 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 পূর্ণপুরি 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);
        }
    }
}
}
}

```

অনুশীলনী

১। নিচের ধারা গুলো for, while, do-while লুপের সাহায্যে প্রিন্ট করতে হবে।

১ ২ ৩ ৪ ৫ ৬ ৭ ...

১ ৩ ৫ ৭ ৯ ১১ ...

২ ৪ ৬ ৮ ১০ ...

১ ২ ৪ ৮ ১৬ ...

২। নিচের প্যাটার্ন গুলো যেকোনো লুপ ব্যবহার করে তৈরি করতে হবে।

*

**

*

* *

* * *

* * * *

* * * * *

1

12

123

1234

12345

1

1 2

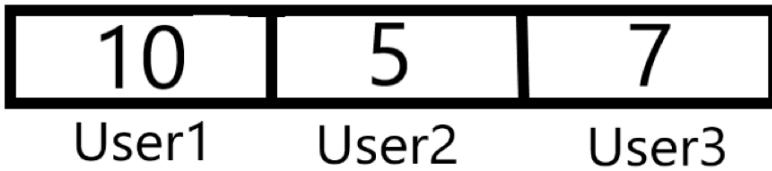
1 2 3

1 2 3 4

1 2 3 4 5

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 ইন্ডেক্সিং ও আছে। Array তে ইন্ডেক্সিং শুরু হয় ০ থেকে। ইন্ডেক্স এর সাহায্যে আমরা ডাটা গুলো পেতে পারি। যেমন: ধরি Array টির নাম User। User array এর ০ ইন্ডেক্স এ আমরা ইউজার ১ এর ডাটা পাবো। অর্থাৎ ১০। এভাবে ইন্ডেক্স ১ এ ৫ এবং ইন্ডেক্স ২ তে ৭ পাবো। উল্লেখ্য, ইন্ডেক্সিং শুরু হয় শূন্য থেকে।

এখন কোড এ চলে যাই।

একটি Array এর ডিক্লেয়ারেশন এরকম:

```
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]+" ");
        }
    }
}
```

Array এর length দেখার জন্য একটি উপায় আছে। এর জন্য যেই array এর length দেখতে চাই সেই array এর নাম লিখে (.) দিয়ে length লিখতে হবে।

test1.java

```
import java.util.Scanner;

public class test1 {

    public static void main(String[] args) {
        int[] arr = {1,2,3,4,5};

        System.out.println(arr.length);

    }
}
```

আউটপুট আসবে 5।

এখন array কে সর্ট করার একটি বিল্ট ইন মেথড দেখি।

test1.java

```
import java.util.Arrays;

public class test1 {

    public static void main(String[] args) {
        int[] arr = {5,3,5,3,1};
        Arrays.sort(arr);
        for (int i = 0; i < arr.length; i++) {
            System.out.print(arr[i]+" ");
        }
    }
}
```

এই বিল্ট ইন Arrays ক্লাসের sort() মেথডটির সাহায্যে আমরা খুব সহজেই একটি array কে ascending ক্রমে সাজাতে পারি। Descending ক্রমে সাজাতে আমাদের নিজেদের কোড করতে হবে। আমরা যখন ArrayList শিখব তখন ArrayList এর descending sort করার বিল্ট ইন মেথড শিখব। Array তে descending sort এর বিল্ট ইন মেথড নেই।

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

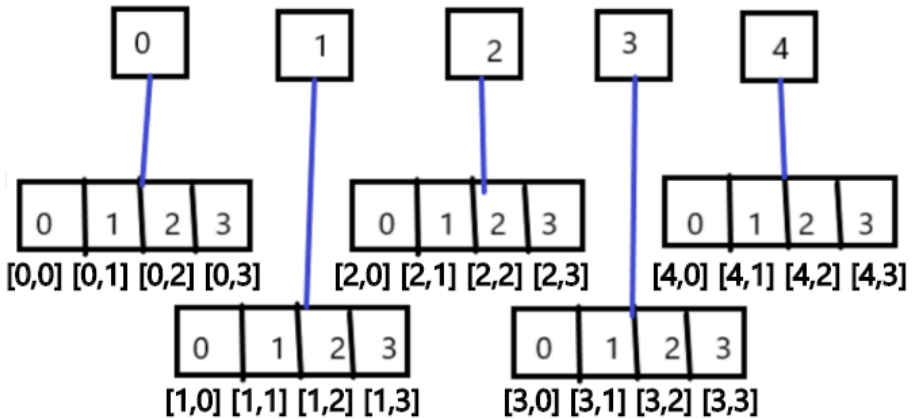
এখন two dimensional array দেখি।

Two dimensional array এর declaration এরকম।

booktest.java

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

নিচের চিত্রটি 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।

booktest.java

```
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।

booktest.java

```
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।

booktest.java

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

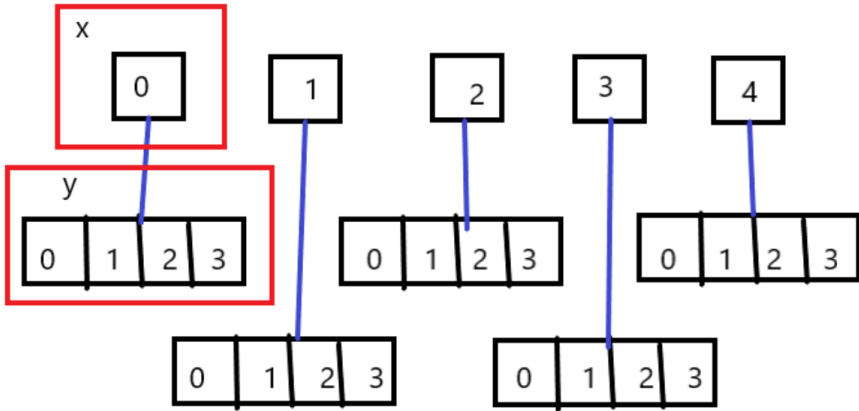
twodarr[0] = new int[3]; //এখানে [0] হচ্ছে index আর [3] এটি হচ্ছে এই index এ যে array টি // রাখলাম তার সাইজ

```
twodarr[1] = new int[4];
twodarr[2] = new int[5];
twodarr[3] = new int[6];
twodarr[4] = new int[7];
```

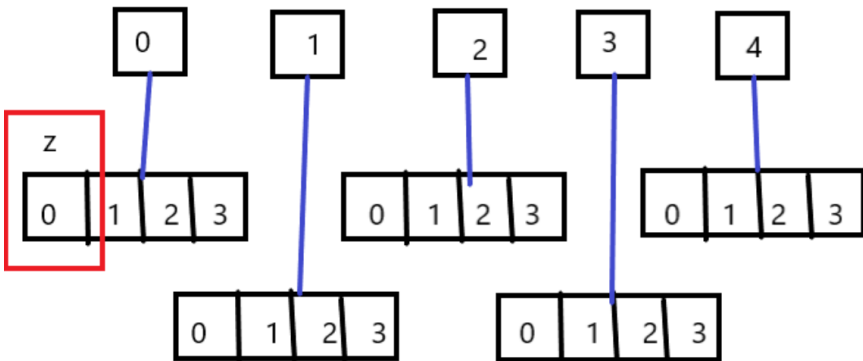
```
}
```

}

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 ।

booktest.java

```
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।

booktest.java

```
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][][];
```

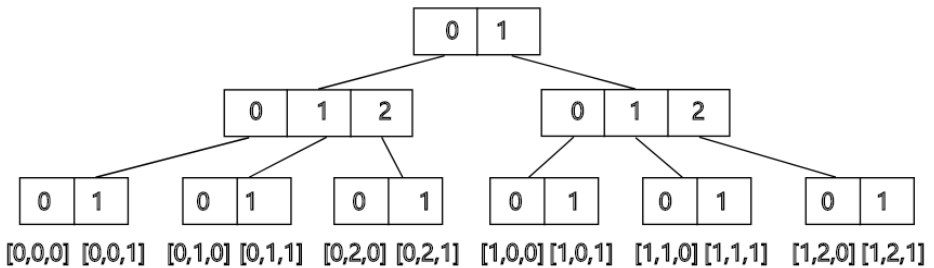
এখন আমরা যদি `arr[n]` এই index এ যাই তাহলে একটি 2-D array রাখতে পারব। যদি `arr[n][n]` এই index পর্যন্ত যাই তাহলে একটি single dimensional array রাখতে পারব। আর যদি `arr[n][n][n]` এই index পর্যন্ত যাই তবে একটি int রাখতে পারব। নিচে উদাহরন দেওয়া হল।

`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।

নিচে চিত্রের মাধ্যমে 3D array দেখানো হল।



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। for-each লুপের আরেক নাম Enhanced for loop।

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।

অনুশীলনী

- ১। ৫,৩,৮,১,৬,৩ এই সংখ্যাগুলো নিয়ে একটি Array তৈরি করতে হবে।
- ২। Array টি থেকে সবচেয়ে বড় এবং সবচেয়ে ছোট সংখ্যাটি খুঁজে বের করতে হবে।
- ৩। Array টির তৃতীয় পদের সংখ্যাটি প্রিন্ট করতে হবে।
- ৪। Array টির চতুর্থ পদের সংখ্যাটি পরিবর্তন করে ২ করে দিতে হবে।
- ৫। Array টিকে সর্ট করতে হবে। Ascending এবং Descending দুইভাবেই। Built-in Arrays.sort() মেথড ব্যবহার করে সর্ট করার পর sort() মেথড ছাড়া নিজে Algorithm তৈরি করে Array টি সর্ট করার চেষ্টা করতে হবে। সার্টিং এর প্রচলিত কিছু Algorithm আছে। যেমনঃ Bubble sort, selection sort, insertion sort, quick sort, heap sort, radix sort ইত্যাদি। এগুলো ব্যবহার করেও করা যেতে পারে।

মেথড

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

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

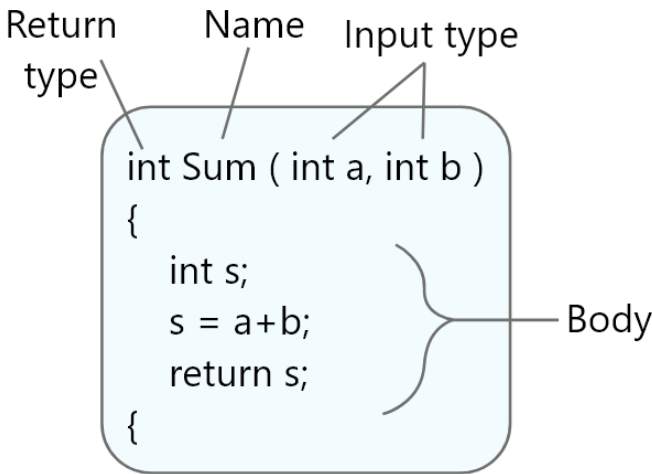


Figure: Method Prototype

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। ইনপুট টাইপ কয়েক ধরনের হতে পারে। যেমন - `int`, `float`, `double`, `char`, `String`, যেকোনো ক্লাসের অবজেক্ট আরও অনেক কিছু। আবার একটি মেথড কোন ইনপুট নাও নিতে পারে।

রিটার্ন টাইপ

মেথড এর কাজ শেষে মেথডটি থেকে কোন ধরনের ডাটা পাওয়া যাবে তাই এখানে বলা থাকে। রিটার্ন টাইপ ও বেশ কয়েক ধরনের থাকতে পারে। যদি কোন মেথড কোন কিছু রিটার্ন না করে তবে তার রিটার্ন টাইপ দিতে হয় void। int, float, double, char, String, যেকোনো ক্লাসের অবজেক্ট আরও অনেক কিছু রিটার্ন টাইপ হতে পারে।

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।

এখনো আমরা ক্লাস এবং অবজেক্ট শিখিনি। ক্লাসের মেথড কল করার জন্য কিছু নিয়ম আছে। একটি ক্লাসের মেথডকে অন্য ক্লাস থেকে কল করার জন্য ওই ক্লাসের একটি অবজেক্ট খুলে নিতে হয়। তারপর ওই অবজেক্ট এর নাম লিখে একটি (.) দিয়ে মেথডটির নাম লিখে () বন্ধনী বা Curly Braces দিতে হয়। যদি মেথডটি আর্গুমেন্ট accept করে তাহলে সেগুলো দিতে হবে। যদি মেথডটি static মেথড হয়, তবে ওই মেথড কল করার জন্য ওই ক্লাসের কোন অবজেক্ট খুলতে হয়না। ক্লাসের সাহায্যেই access করা যায়।

অনুশীলনী

- ১। এমন একটি মেথড তৈরি করতে হবে যা আপনার নাম প্রিন্ট করবে।
- ২। এমন একটি মেথড তৈরি করতে হবে যা দুইটি সংখ্যার মধ্যে বড় সংখ্যাটি রিটার্ন করে।
- ৩। এমন একটি মেথড তৈরি করতে হবে যা একটি character বড় হাতের নাকি ছোট হাতের যাচাই করবে। এর রিটার্ন টাইপ হবে boolean।

Math ক্লাসের কিছু মেথড

কিছু গাণিতিক কাজ সহজে করার জন্য Math ক্লাসের কিছু বিল্ট ইন মেথড রয়েছে। তার কয়েকটি নিচে দেওয়া হল। এগুলো আমাদের প্রোগ্রামিং কে আরও সহজ করে দেয়। এই মেথডগুলো static মেথড, তাই এখানে অবজেক্ট খোলার প্রয়োজন নেই।

মেথড	বর্ণনা	উদাহরণ
Math.abs()	এটি absolute value রিটার্ন করে। অর্থাৎ ঋণাত্মক সংখ্যা ইনপুট দিলে ধনাত্মক সংখ্যা রিটার্ন করে। আর ধনাত্মক সংখ্যা ইনপুট দিলে ধনাত্মক সংখ্যাই রিটার্ন করে।	<pre>double x = -11.23; double y = 34.12; System.out.println(Math.abs(x));</pre> <p>আউটপুট: 11.23</p>
Math.max()	দুইটি সংখ্যার মধ্যে বড় সংখ্যাটি রিটার্ন করে।	<pre>double x = -11.23; double y = 34.12; System.out.println(Math.max(x,y));</pre> <p>আউটপুট: 34.12</p>
Math.min()	দুইটি সংখ্যার মধ্যে ছোট সংখ্যাটি রিটার্ন করে।	<pre>double x = -11.23; double y = 34.12; System.out.println(Math.min(x,y));</pre> <p>আউটপুট: -11.23</p>

Math.round()	এটি কোন একটি দশমিক সংখ্যার নিকটবর্তী পূর্ণসংখ্যা রিটার্ন করে।	System.out.println(Math.round(3.456)); আউটপুট: 3
Math.sqrt()	একটি সংখ্যার বর্গমূল রিটার্ন করে। এর রিটার্ন টাইপ double।	System.out.println(Math.sqrt(3.456)); আউটপুট: 1.85903200617956
Math.cbrt()	একটি সংখ্যার ঘনমূল রিটার্ন করে। এর রিটার্ন টাইপ double।	System.out.println(Math.cbrt(3.456)); আউটপুট: 1.5119052598738478
Math.pow()	এই ফাংশনটি প্রথম আর্গুমেন্টকে দ্বিতীয় আর্গুমেন্টের সূচকে পরিণত করে রিটার্ন করে। এর রিটার্ন টাইপ double।	System.out.println(Math.pow(3.456,4)); আউটপুট: 142.657607172096
Math.ceil()	এটি আর্গুমেন্ট এর চেয়ে বড় বা সমান ক্ষুদ্রতম পূর্ণসংখ্যার মান খুঁজে পেতে ব্যবহৃত হয়। এর রিটার্ন টাইপ double।	System.out.println(Math.ceil(3.456)); আউটপুট: 4.0
Math.floor()	এটি বৃহত্তম পূর্ণসংখ্যার মান খুঁজে বের করতে ব্যবহৃত হয় যা আর্গুমেন্টের থেকে ছোট বা সমান। এর রিটার্ন টাইপ double।	System.out.println(Math.floor(3.456)); আউটপুট: 3.0
Math.random()	এটি শূন্য থেকে এক এর মধ্যে একটি random সংখ্যা রিটার্ন	System.out.println(Math.random());

	<p>করে। এর রিটার্ন টাইপ double। কোন একটি নির্দিষ্ট রেঞ্জ এর মধ্যে random সংখ্যা পাওয়ার জন্য এই সূত্রটি ব্যবহার করতে পারি।</p> $\text{Min} + (\text{Math.random()} * (\text{Max} - \text{Min}))$ <p>ইন্টিজার দরকার হলে আমরা type casting করে নিতে পারি।</p>	<p><code>System.out.println(10+Math.random()*(100-10));</code></p> <p>আউটপুট: এটি এক এক সময় এক এক মান রিটার্ন করবে।</p>
Math.log()	এটি একটি double সংখ্যার স্বাভাবিক লগারিদম প্রদান করে।	<p><code>System.out.println(Math.log(3.45));</code></p> <p>আউটপুট: 1.2383742310432684</p>
Math.log10()	এটি একটি double সংখ্যার ১০ ভিত্তিক লগারিদম প্রদান করে।	<p><code>System.out.println(Math.log10(3.45));</code></p> <p>আউটপুট: 0.5378190950732742</p>
Math.exp()	এটি e কে আর্গুমেন্ট এর সূচকে নিয়ে একটি double সংখ্যা রিটার্ন করে।	<p><code>System.out.println(Math.exp(3.45));</code></p> <p>আউটপুট: 31.500392308747937</p>
Math.sin()	এটি একটি প্রদত্ত double সংখ্যার ত্রিকোণমিতিক সাইন মান রিটার্ন ব্যবহৃত হয়।	<p><code>System.out.println(Math.sin(Math.PI/4));</code></p> <p>আউটপুট: 0.7071067811865475</p>

Math.cos()	এটি একটি প্রদত্ত double সংখ্যার ত্রিকোণমিতিক কোসাইন মান রিটার্ন ব্যবহৃত হয়।	System.out.println(Math.cos(Math.PI/4)); আউটপুট: 0.7071067811865476
Math.tan()	এটি একটি প্রদত্ত double সংখ্যার ত্রিকোণমিতিক স্পর্শক মান রিটার্ন ব্যবহৃত হয়।	System.out.println(Math.tan(Math.PI/4)); আউটপুট: 0.9999999999999999
Math.asin()	এটি একটি প্রদত্ত double সংখ্যার ত্রিকোণমিতিক সাইন ইনভার্স মান রিটার্ন ব্যবহৃত হয়।	System.out.println(Math.asin(0.5)); আউটপুট: 0.5235987755982989
Math.acos()	এটি একটি প্রদত্ত double সংখ্যার ত্রিকোণমিতিক কোসাইন ইনভার্স মান রিটার্ন ব্যবহৃত হয়।	System.out.println(Math.acos(0.5)); আউটপুট: 1.0471975511965979
Math.atan()	এটি একটি প্রদত্ত double সংখ্যার ত্রিকোণমিতিক ট্যানজেন্ট ইনভার্স মান রিটার্ন ব্যবহৃত হয়।	System.out.println(Math.atan(0.5)); আউটপুট: 0.4636476090008061

অনুশীলনী

১। নিচের বীজগণিতিক রাশিগুলো Math ক্লাসের মেথডের মাধ্যমে প্রকাশ করতে হবে।

$$5 \cdot 6^9 + \sqrt{25}$$

$$\sqrt{(25^5) + \sin \pi/3}$$

২। 1400 বর্গ একক ক্ষেত্রফল বিশিষ্ট একটি মেঝে ইট দিয়ে বাঁধাই করতে $৫*৬$ বর্গ একক ক্ষেত্রফল বিশিষ্ট কতগুলো ইট লাগবে বের করতে হবে। এখানে ইটের সংখ্যা অবশ্যই পূর্ণসংখ্যা হবে।

৩। একটি প্যাকেটে ধরা যাক ৫০ টি চিপস থাকে। এখন ৩২৪৫ টি চিপস দিয়ে কতগুলো প্যাকেট ভরা যাবে তা বের করার একটি প্রোগ্রাম লিখতে হবে।

স্ট্রিং

স্ট্রিং হচ্ছে ক্যারেক্টার এর অ্যারে। আমরা char আর Character এ মাত্র একটি অক্ষর রাখতে পারতাম। এখন যদি দরকার হয় একটি শব্দকে রাখার অথবা একটি বাক্যকে অথবা একটি প্যারাগ্রাফকে, তখন কি করবো? এর সমাধান হচ্ছে স্ট্রিং।

স্ট্রিং কে double quotation (“ ”) এর ভিতরে রাখতে হয়। স্ট্রিং এর ডিক্লারেশন দেখি। স্ট্রিং যেহেতু রেফারেন্স টাইপ ডাটা টাইপ সেহেতু এটিকে এভাবে ডিক্লেয়ার করার কথা।

```
String str = new String("Abduz Zami");
System.out.println(str);
```

এভাবে না করে আমরা আরও সহজে স্ট্রিং ডিক্লেয়ার করতে পারি। এটা প্রিমিটিভ টাইপের ডিক্লারেশনের মত।

```
String str = "Abduz Zami";
System.out.println(str);
```

ইউজার এর কাছে থেকে স্ট্রিং ইনপুট নেওয়া দেখি এখন।

```
String str;
Scanner scanner = new Scanner(System.in);
str = scanner.next();
```

next() মেথডটি স্পেস পাওয়ার আগ পর্যন্ত রিড করে।
আরও একটি মেথড আছে স্ট্রিং স্ক্যান করার।

```
String str1;
str1 = scanner.nextLine();
System.out.println(str1);
```

nextLine() মেথডটি নতুন লাইন পাওয়ার আগ পর্যন্ত স্ক্যান করে।

`next()` এবং `nextLine()` নিয়ে ইতঃপূর্বে বিস্তারিত আলোচনা করা হয়েছে। “ইউজার থেকে ইনপুট নেওয়া” অধ্যায়ে।

আমরা স্ট্রিং এর ডিক্লেয়ারেশন এবং ইউজার ইনপুট নেওয়া শিখলাম। এখন স্ট্রিং সম্পর্কে একটি গুরুত্বপূর্ণ কথা বলব। সেটি হল স্ট্রিং কে বলা হয় ইম্মিউটেবল (Immutable) ডাটা টাইপ। এর কারণটি হল এই যে, স্ট্রিং ভেরিএবল একবার ইনিশিয়ালাইজ করলে পরবর্তীতে আর তার কোন পরিবর্তন করা যায় না। বিষয়টা কোডের মাধ্যমে দেখাই।

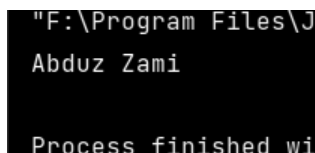
test1.java

```
import java.util.Scanner;

public class test1 {

    public static void main(String[] args) {
        String str = "Abduz Zami";
        str.concat("Abdus Sami");
        System.out.println(str);
    }
}
```

এই কোডে `concat` নামে একটি ফাংশন ব্যবহার করা হয়েছে। এর কাজ হল একটি স্ট্রিং এর সাথে অন্য একটি স্ট্রিং কে সংযুক্ত করা। তো আউটপুটে আসার কথা ছিল `Abduz ZamiAbdusSami`। কিন্তু আউটপুটে এসেছে এটি।



```
"F:\Program Files\J
Abduz Zami
Process finished wi
```

আরেকটি উদাহরণ দেখি।

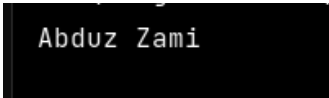
Test1.java

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

```
public class test1 {

    public static void main(String[] args) {
        String str = "Abduz Zami";
        str.toLowerCase();
        System.out.println(str);
    }
}
```

এখানে toLowerCase() মেথডটি ব্যবহার করা হয়েছে যার কাজ স্ট্রিং এর সব character কে ছোট হাতের করে দেওয়া। এটার আউটপুট দেখলেও দেখব যে মূল স্ট্রিং এর কোন পরিবর্তন হয়নি।



Immutable স্ট্রিং এর এই সমস্যার সমাধান কয়েকভাবে করা যায়। তার মধ্যে একটি হল বিকল্প ডাটা টাইপ এর ব্যবহার - `StringBuilder` আর `StringBuffer`। এগুলো আমরা পরবর্তীতে শিখব। অন্য উপায়টি হল স্ট্রিংটিতে কোন পরিবর্তন আনার লাইনেই ওই স্ট্রিং এ অ্যাসাইন করে দেওয়া। কোডের মাধ্যমে দেখি।

test1.java

```
import java.util.Scanner;

public class test1 {

    public static void main(String[] args) {
        String str = "Abduz Zami";
        str = str.toLowerCase();
        System.out.println(str);
    }
}
```

এর আউটপুট আসবে এটা।

```
"F:\Program Fil  
abduz zami
```

এবার ঠিকঠাক আউটপুট এসেছে।

শুরুতে বলেছিলাম স্ট্রিং হচ্ছে character এর array। কিন্তু স্ট্রিং এ array এর মত ইনডেক্স ওয়াইজ value পরিবর্তন করা যায়না। তবে যেকোনো ইনডেক্স এর character টি পাওয়া যায়। অর্থাৎ read করতে কোন সমস্যা নেই। যত জটিলতা সব write করতে। আমাদের ইনডেক্স ওয়াইজ কাজ হরহামেশাই করতে হয় সেজন্য একটি কাজ করা যায়। সেটি হল স্ট্রিং কে শুরুতেই char এর array তে নিয়ে নেওয়া। কাজটি আমরা এভাবে করতে পারি।

test1.java

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

```
public class test1 {
```

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

```
        String str = "Abduz Zami";
```

```
        char[] carr = str.toCharArray();
```

```
        System.out.println(carr[0]);
```

```
        System.out.println(carr[1]);
```

```
        carr[0] = 'X';
```

```
        carr[1] = 'Y';
```

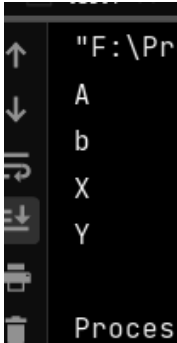
```
        System.out.println(carr[0]);
```

```
        System.out.println(carr[1]);
```

```
    }
```

```
}
```

এর আউটপুটটি নিম্নরূপ।



এখানে আমরা বিভিন্ন index এর value দেখতে পারছি এবং তা পরিবর্তন ও করতে পারছি। যেমন ০ এবং ১ নম্বর ইনডেক্স এ যথাক্রমে A , b ছিল যাকে পরিবর্তন করে আমরা X ও Y করেছি। Array নিয়ে আগের অধ্যায়ে বিস্তারিত আলোচনা করা হয়েছে।

স্ট্রিং এর বেশ কয়েকটি মেথড বা ফাংশন আছে যা প্রচুর ব্যবহার হয়। তার কয়েকটি নিচে দেওয়া হল।

String ক্লাসের কিছু মেথড

নিচে একটি স্ট্রিং নিয়ে তার উপর ভিত্তি করে টেবিল এ সব গুলো উদাহরণ সহ দেখাবো। এই মেথডগুলো non-static মেথড হওয়ায় এদের কে অবজেক্ট খুলে অবজেক্টের মাধ্যমে কল করতে হয়।

```
String str = "MATRIVASAY JAVA SHIKHI";
```

মেথডের নাম	বর্ণনা	উদাহরণ
length()	এর সাহায্যে স্ট্রিং এর সাইজ জানা যায়। এটি একটি int রিটার্ন করে।	<pre>int len = str.length(); System.out.println(len);</pre> <p>Output:</p>

		22
toLowerCase()	এর সাহায্যে স্ট্রিং এর সব character কে ছোট হাতের করা যায়। এটি একটি String রিটার্ন করে।	<pre>String stL = str.toLowerCase(); System.out.println(stL);</pre> <p>Output: matrivasay java shikhi</p>
charAt()	এর সাহায্যে কোন নির্দিষ্ট ইনডেক্স এর character খুঁজে নেওয়া যায়। এটি char রিটার্ন করে।	<pre>System.out.println(str.charAt(5));</pre> <p>Output: V</p>
substring()	এর সাহায্যে কোন নির্দিষ্ট সীমার ভিতরে স্ট্রিং এর অংশ পাওয়া যায়। যেমন এখানে ২ থেকে ৫ এর আগ পর্যন্ত স্ট্রিং এর অংশ পাওয়া গিয়েছে। এটি একটি String রিটার্ন করে।	<pre>System.out.println(str.substring(2,5));</pre> <p>Output: TRI</p>
toUpperCase()	এর সাহায্যে স্ট্রিং এর সব character কে বড় হাতের করা যায়। এটি একটি String রিটার্ন করে।	<pre>System.out.println(str.toUpperCase());</pre> <p>Output: MATRIVASAY JAVA SHIKHI</p>
concat()	এর সাহায্যে একটি স্ট্রিং এর সাথে আরেকটি স্ট্রিং যুক্ত করা যায়। এটি একটি String রিটার্ন করে।	<pre>System.out.println(str.concat(" - ABDUZ ZAMI"));</pre> <p>Output: MATRIVASAY JAVA SHIKHI - ABDUZ ZAMI</p>

contains()	এর সাহায্যে যাচাই করা যায় যে স্ট্রিং এ কোন নির্দিষ্ট স্ট্রিং আছে কিনা। এটি একটি boolean রিটার্ন করে। অর্থাৎ true/false।	System.out.println(str.contains("JAVA")); Output: true
isEmpty()	এর সাহায্যে দেখা যায় স্ট্রিং টি ফাকা কিনা। এটি একটি boolean রিটার্ন করে।	System.out.println(str.isEmpty()); Output: false
equals()	এর সাহায্যে দেখা যায় স্ট্রিং টি অন্য স্ট্রিং এর অনুরূপ কিনা। এটি একটি boolean রিটার্ন করে।	System.out.println(str.equals("ENGLISH E JAVA SHIKHI")); Output: false
compareTo()	এর সাহায্যে lexicographically compare করা হয় দুইটি স্ট্রিং কে। আমরা জানি প্রতিটি character এর ই আসকি কোড আছে। এই আসকি কোডের পার্থক্যের মাধ্যমে এই তুলনাটি করা হয়। এটি একটি int রিটার্ন করে।	System.out.println(str.compareTo("SHIKHBONA")); Output: -6
endsWith()	এর সাহায্যে দেখা যায় স্ট্রিং টি নির্দিষ্ট স্ট্রিং দিয়ে শেষ হচ্ছে কিনা। এটি একটি boolean রিটার্ন করে।	System.out.println(str.endsWith("SHIKHI")); Output: true

startsWith()	এর সাহায্যে দেখা যায় স্ট্রিং টি নির্দিষ্ট স্ট্রিং দিয়ে শুরু হচ্ছে কিনা। এটি একটি boolean রিটার্ন করে।	<pre>System.out.println(str.startsWith("M"));</pre> <p>Output: true</p>
indexOf()	এটি একটি character এর ইনডেক্স বলে দেয়। আর যদি খুঁজে না পায় তবে -1 রিটার্ন করে। এটি ইন্টিজার রিটার্ন করে।	<pre>System.out.println(str.indexOf('V'));</pre> <p>Output: 5</p> <pre>System.out.println(str.indexOf("JAVA"));</pre> <p>Output: 11</p>
isBlank()	স্ট্রিং টি ফাকা কিনা দেখা যায়। এটি বুলিয়ান রিটার্ন করে।	<pre>System.out.println(str.isBlank());</pre> <p>Output: false</p>
lastIndexOf()	এটি একটি character বা স্ট্রিং কে শেষ দিক থেকে খুঁজে ইনডেক্স বের করে দেয়। আর যদি খুঁজে না পায় তবে -1 রিটার্ন করে।	<pre>System.out.println(str.lastIndexOf('A'));</pre> <p>Output: 14</p> <pre>System.out.println(str.lastIndexOf("JAVA"));</pre>

		Output: 11
replace()	এটি স্ট্রিং এর একটি character কে অন্য একটি character দিয়ে পরিবর্তন করে দেয়। এটি একটি String রিটার্ন করে।	System.out.println(str.replace('A','Z')); Output: MZTRIVZSZY JZVZ SHIKHI
replaceAll()	এটি স্ট্রিং এর একটি অংশকে অন্য একটি স্ট্রিং দিয়ে পরিবর্তন করে দেয়। এটি একটি String রিটার্ন করে।	System.out.println(str.replaceAll("JAVA","PYTHON")); Output: MATRIVASAY PYTHON SHIKHI
split()	এটি একটি character এর ভিত্তিতে একটি স্ট্রিং কে ভাগ করে একটি স্ট্রিং এর array তে পরিণত করে। এই character টি যেকোনো বর্ণ বা সানস্কেতিক চিহ্ন হতে পারে। একাধিক character এর ভিত্তিতেও এ কাজটি করা যায়।	String[] splittedArray = str.split(" "); for (int i = 0; i < splittedArray.length; i++) { System.out.print(splittedArray[i] + " "); } Output: MATRIVASAY JAVA SHIKHI

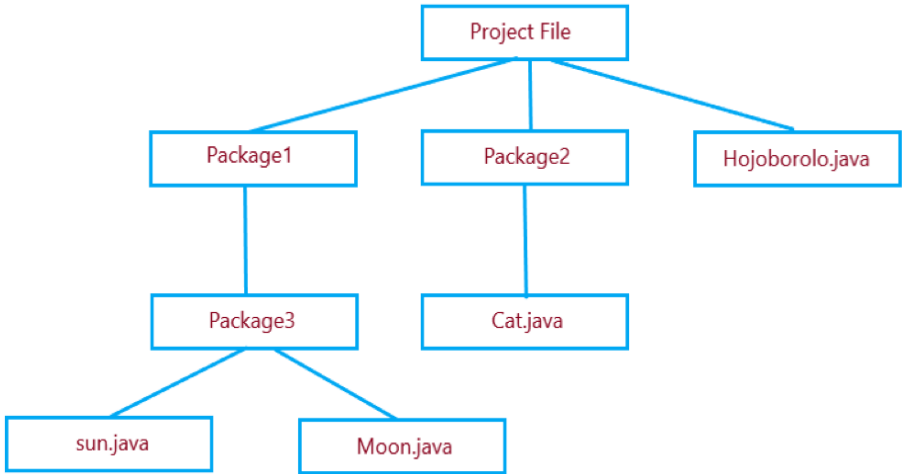
		<pre>String[] splittedArray = str.split("[MSI,.]"); for (int i = 0; i < splittedArray.length; i++) { System.out.print(splittedArray[i] +" "); } Output: ATR VA AY JAVA H KH MATRIVASAY JAVA SHIKHI</pre>
trim()	এটি স্ট্রিং এর উভয় দিক হতে whitespace গুলো সরিয়ে দেয় এবং ফলাফল স্ট্রিং হিসেবে রিটার্ন করে।	<pre>str = " MATRIVASAY JAVA SHIKHI "; System.out.println(str.trim()); Output: MATRIVASAY JAVA SHIKHI</pre>
toCharArray()	এটি স্ট্রিং কে character এর Array তে পরিবর্তন করে দেয়।	<pre>char[] carr = str.toCharArray(); for (int i = 0; i < carr.length; i++) { System.out.print(carr[i]+" "); } Output: MATRIVASAY JAVA SHIKHI</pre>

উপরে প্রয়োজনীয় প্রায় সবগুলো মেথড এর উদাহরণ সহ বর্ণনা দেওয়া হয়েছে। এগুলো শুধু দেখে গেলেই হবেনা। নিজে নিজে কোড করে দেখতে হবে।

অনুশীলনী

- ১। একটি স্ট্রিং ইউজার ইনপুট নিয়ে তাতে কতগুলো vowel আছে তা গণনা করে প্রিন্ট করতে হবে।
- ২। একটি স্ট্রিং কে char এর array তে রূপান্তর করতে হবে।
- ৩। একটি স্ট্রিং এর সাথে আরেকটি স্ট্রিং যোগ করে প্রিন্ট করতে হবে।
- ৪। একটি স্ট্রিং এর একটি নির্দিষ্ট ইনডেক্স থেকে অন্য একটি ইনডেক্স এর আগ পর্যন্ত প্রিন্ট করতে হবে।

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 সে ক্ষেত্রে ওই ক্লাস গুলোকে একই প্যাকেজের ভিতরে রাখলে পরবর্তীতে আগ্রহেড 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) {

    }
}
```

প্যাকেজ ইমপোর্ট করা

আমাদের অনেক সময় অন্য প্যাকেজের ক্লাসের প্রোপার্টি বা মেথড ব্যবহার করতে হয়। সেজন্য ওই প্যাকেজটি ইমপোর্ট করতে হয়। প্যাকেজ ইমপোর্ট এর জন্য জাভা ফাইলের একেবারে উপরে কিছু কথা লিখতে হয়। আমরা Scanner দিয়ে ইউজার ইনপুট নেই। যা এই বইয়ে বেশ কয়েকবার দেখেছি। সেটিই আবার দেখি।

test1.java

```
package package1;
import java.util.Scanner;

public class test1 {

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

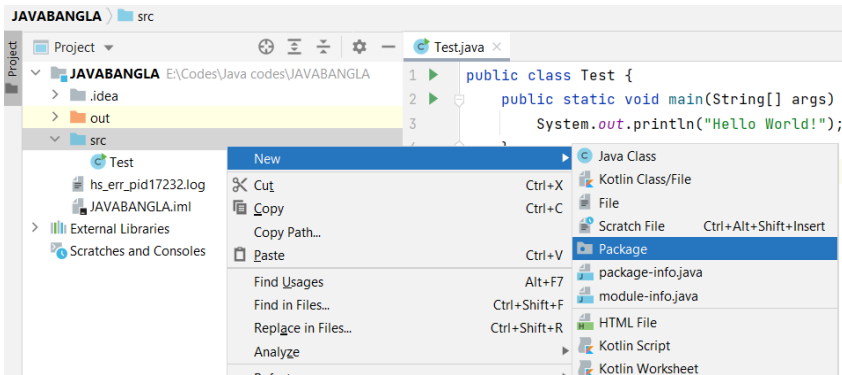
এখানে import java.util.Scanner; এই লাইনটি ইমপোর্ট এর কাজ করছে। এই লাইনটির মানে হচ্ছে java প্যাকেজের util সাব প্যাকেজে যে Scanner ক্লাসটি রয়েছে তাকে ইমপোর্ট করা হয়েছে।

আমি যদি এভাবে লিখতাম- import java.util.*;

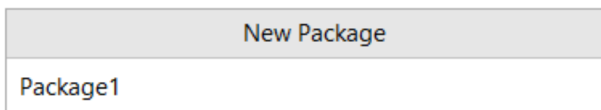
তাহলে util প্যাকেজের সব গুলো ক্লাস ইমপোর্ট হয়ে যেত। অর্থাৎ * এর মাধ্যমে সব ক্লাস ইমপোর্ট করা যায়।

IDE তে প্যাকেজ খোলার নিয়ম

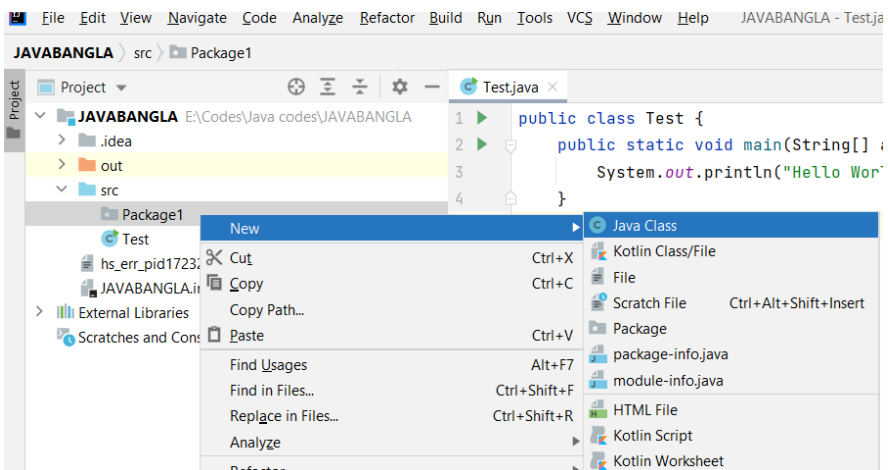
প্যাকেজ খোলার জন্য src ফোল্ডার এ মাউস দিয়ে রাইট ক্লিক করলে একটি উইন্ডো আসবে। সেখানে new তে ক্লিক করলে আরেকটি উইন্ডো আসবে। সেখানে Package এ ক্লিক করতে হবে।



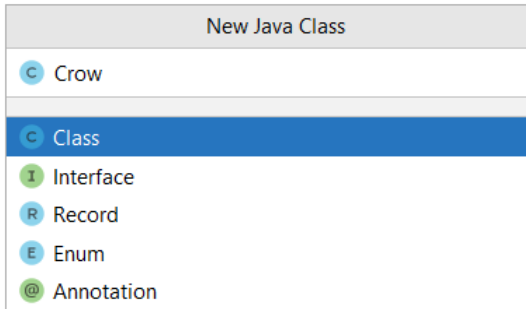
তারপর আরেকটি উইন্ডো আসবে সেখানে প্যাকেজের নাম দিতে হবে। নাম লিখে এন্টার দিলেই একটি প্যাকেজ তৈরি হয়ে যাবে।



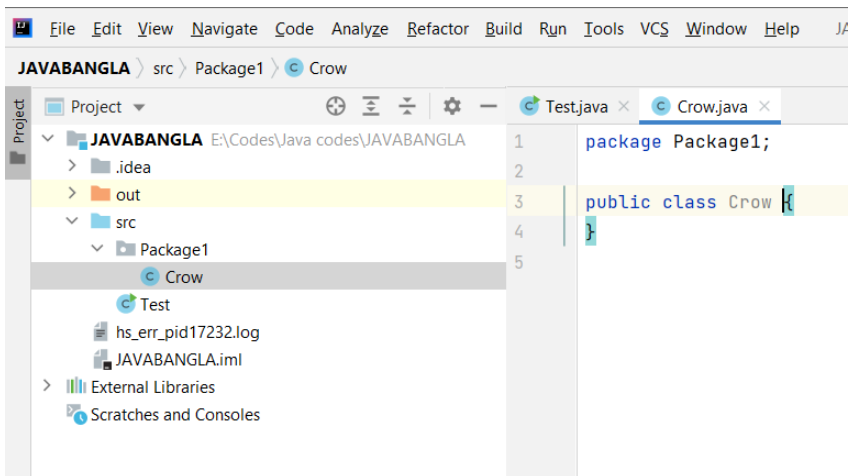
প্যাকেজ খোলার পর ওই প্যাকেজের উপর মাউসের রাইট ক্লিক করলে একটি উইন্ডো আসবে। সেখানে new তে ক্লিক করলে আরেকটি উইন্ডো আসবে। সেখানে Java Class এ ক্লিক করলে আরও একটি উইন্ডো আসবে।



এখানে জাভা ক্লাসের নাম লিখে নিচে Class অপশনটি সিলেক্টেড থাকা অবস্থায় এন্টার দিলে ওই নামে একটি জাভা ক্লাস তৈরি হয়ে যাবে।



ক্লাস খোলার পর এরকম দেখা যাবে। এখানে Curley Braces বা দ্বিতীয় বন্ধনীর { } ভিতরে কোড লিখতে হবে।



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।

প্রতিটি ক্লাসে কিছু variable এবং মেথড থাকে। আমরা একটি Car ক্লাস তৈরি যার brand, model, speed, color এই চারটি variable থাকবে এবং Repaint() নামে একটি মেথড থাকবে। package1 এর ভিতরে আমরা Car নামে একটি ক্লাস নেই।

Car.java

```
package package1;

public class Car {
    String model, brand, color;
    double speed;

    void Repaint(String tocolor){
        color = tocolor;
    }
}
```

এখানে দেখানো হয়েছে কিভাবে একটি ক্লাসের variable এবং method ডিক্লেয়ার করতে হয়।

এখন আমরা অবজেক্ট তৈরি করব। এর জন্য একই প্যাকেজ এ আরেকটি ক্লাস খুলি। নাম দেই test1.java।

test1.java

```
package package1;

public class test1 {
```

```

public static void main(String[] args) {
    Car car1 = new Car();
    car1.model = "Corola";
    car1.brand = "Toyota";
    car1.speed = 150.75;
    car1.color = "Blue";
    System.out.println("Color before : "+car1.color);
    car1.Repaint("Black");
    System.out.println("Color after : "+car1.color);
}
}

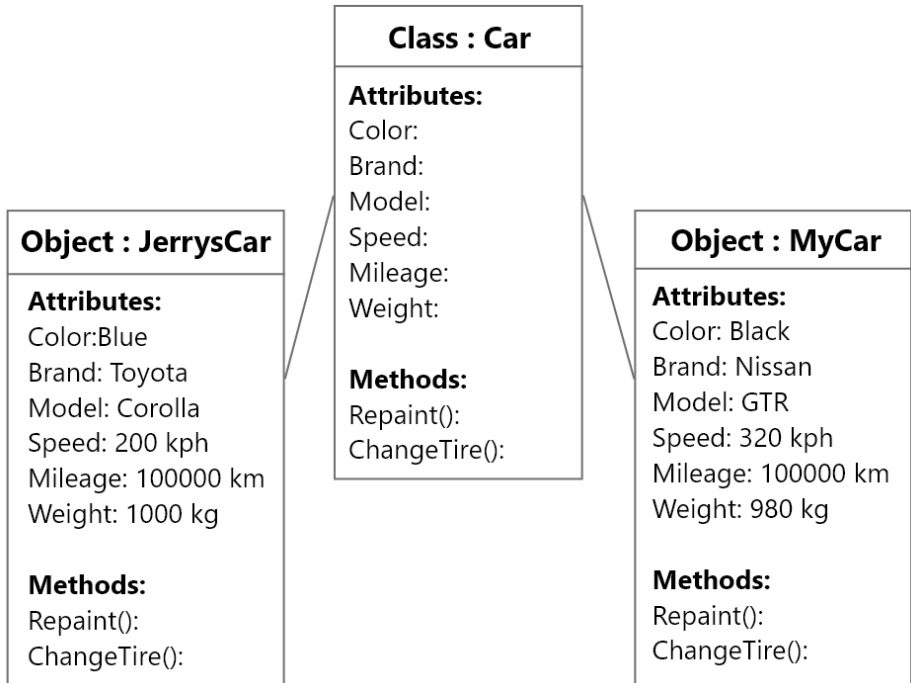
```

এভাবে আমরা Car এর একটি অবজেক্ট তৈরি করতে পারি। এখানে car হচ্ছে reference variable। আগে বলেছিলাম class হচ্ছে একটি reference type data type। new দিয়ে Car এর অবজেক্ট declare করা হয়েছে এবং মেমোরি তে এই অবজেক্ট এর জন্য কিছু জায়গা বরাদ্দ হয়েছে। Object declaration নিয়ে কন্সট্রাক্টর অধ্যায়ে আরও বিস্তারিত বলব। কারণ অবজেক্ট তৈরি করতে কন্সট্রাক্টর এর প্রয়োজন হয়। বলে রাখি এখানে default constructor কল হয়েছে। কোন constructor তৈরি করে না দিলে default constructor টি কল হয়।

এরপরে ওই অবজেক্ট এর জন্য ক্লাসের প্রোপার্টি বা variable গুলোর মান বসিয়ে দেওয়া হয়েছে। এর জন্য প্রথমে অবজেক্ট এর নাম তার পর (.) দিয়ে variable এর নাম লিখতে হয়। এর পর = দিয়ে value লিখতে হয়।

কিভাবে variable এর মান বসাতে হয় তা এখানে দেখলাম আমরা। তারপর আমরা আমাদের তৈরি করা Repaint() মেথড কে কল করেছি। এর জন্য প্রথমে অবজেক্ট এর নাম তার পর (.) দিয়ে মেথডের নাম লিখতে হয়। এর পর () এর ভিতরে আর্গুমেন্ট গুলো পাস করতে হয়। আর্গুমেন্ট মানে হচ্ছে একটি মেথড যা গ্রহণ করে। মেথডের ভিতরে আমরা car এর color variable এর value বদলে দিয়েছি যা পরের লাইনে আউটপুটের মাধ্যমে দেখতে পারছি।

নিচের চিত্রে ক্লাস এবং অবজেক্ট এর একটি চিত্র দেখানো হল।



অনুশীলনী

১। Bike নামে একটি ক্লাস তৈরি করতে হবে যাতে নিচের প্রোপার্টি এবং মেথডগুলো থাকবে। এখানে ইচ্ছা মত রিটার্ন এবং আউটপুট টাইপ নেওয়া যাবে।

Variables	Methods
Brand Model Color Mileage Weight Tyres Brakes Lights Fuel	rePaint() changeTyres() changeBreaks() turnOnOffLights() reFuel()

২। উপরের ক্লাসটি তৈরি করে তার কতগুলো অবজেক্ট তৈরি করতে হবে।

৩। অবজেক্ট গুলোতে value initialize করতে হবে। যেমন: color, brand, mileage ইত্যাদি এসব ইনপুট করতে হবে।

৪। অবজেক্ট গুলোর জন্য মেথডগুলো কল করতে হবে।

রেফারেন্স টাইপের মেমোরি বন্টন

আমরা যদি রেফারেন্স টাইপ এর ক্ষেত্রে মেমোরি allocation দেখতে চাই।

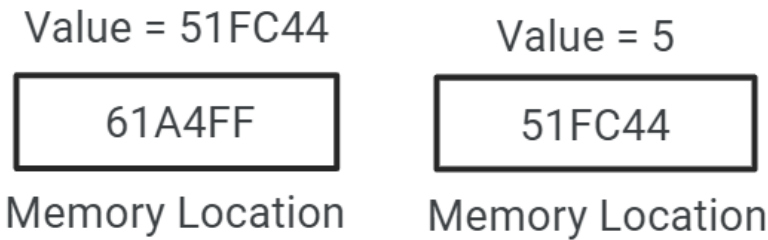
একটি নন-প্রিমিটিভ বা রেফারেন্স টাইপ variable ডিক্লেয়ার করি,

```
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 সেই 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 ৪ 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,

Vehicle.java

```

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।

Main.java

```

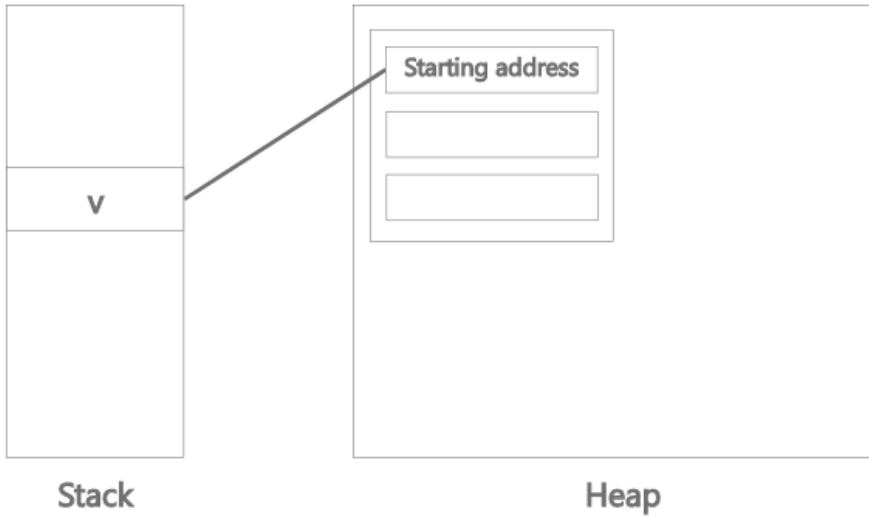
public class Main {

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

```

নিচের চিত্রটি লক্ষ 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 চিহ্নটি বোঝাতে চাই,
 এখানে v হচ্ছে Vehicle এর একটি রেফারেন্স variable যা Vehicle এর অবজেক্ট কে রেফার করছে।
 Vehicle class এর object এর জন্য কি পরিমান 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);
    }

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

Animal class টিতে name এর জন্য একটি String, কতগুলি পা আছে তা 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।

Main.java

```
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 মেথদের 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

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() {
    }
}

```

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

Animal.java

```

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।

Main.java

```
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।

Main.java

```
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

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;
    }

    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। সেক্ষেত্রে 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।

Animal.java

```
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।

অনুশীলনী

১। ধরে নেই Bike নামের একটি ক্লাসের নিম্নোক্ত প্রোপার্টি এবং মেথড আছে।

Variables	Methods
Brand Model Color Mileage Weight Tyres Brakes Lights Fuel	rePaint() changeTyres() changeBreaks() turnOnOffLights() reFuel()

এখন এর জন্য একটি default constructor এবং একটি parameterized constructor তৈরি করতে হবে।

২। তৈরি করা কন্সট্রাক্টর দিয়ে অবজেক্ট তৈরি করতে হবে।

ফাইনালাইজ মেথড

জাতায় 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);
    }

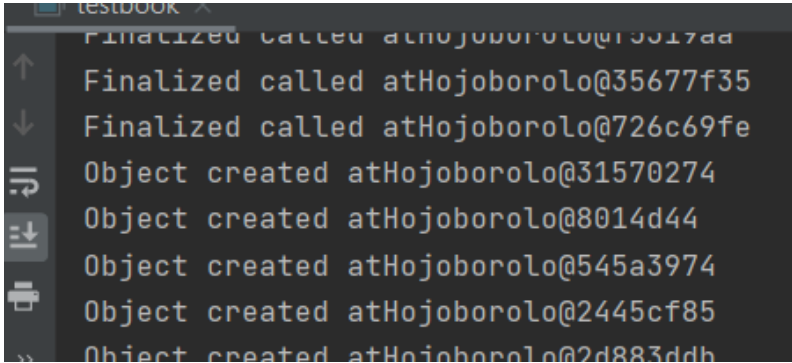
    @Override
    protected void finalize() {
        System.out.println("Finalized called at"+this);
    }
}

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@3317aa
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।

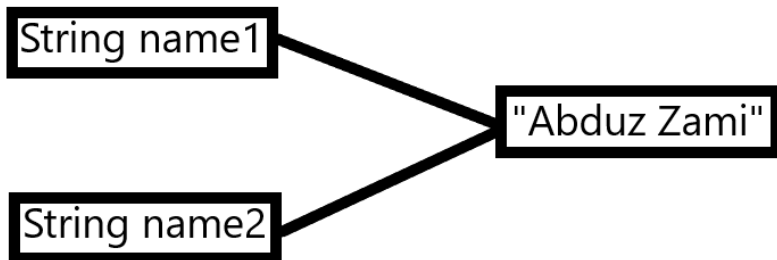
উপরের কোডটি কিছু পরিবর্তন করি

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 মেথডটি তে 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 গ্রহণ 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 তা 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 তা 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। কারণ 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 কারণ 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।

Wrapper ক্লাস গুলোর কিছু মেথড আছে যা আমাদের প্রোগ্রামিংকে সহজ করে দেয়। এর মধ্যে কতগুলো নিচে দেখানো হল।

Integer এর একটি মেথড হচ্ছে `parseInt()`। এর সাহায্যে স্ট্রিং কে ইন্টিজারে রূপান্তর করা যায়। তবে স্ট্রিং এ সব character সংখ্যা হতে হবে।

```
String snum = "12345";
int num = Integer.parseInt(snum);
System.out.println(num);
```

Double এর একটি মেথড হচ্ছে `parseDouble()`। এর সাহায্যে স্ট্রিং কে double এ রূপান্তর করা যায়।

```
String snum = "123.45";
double num = Double.parseDouble(snum);
System.out.println(num);
```

এরকম মেথড Byte, Short, Long, Float এর জন্যেও আছে।

আবার int, short, byte, long, float, double কে স্ট্রিং এও রূপান্তর করা যায়। এর জন্য ব্যবহার করতে হবে String ক্লাসের `valueOf()` মেথডকে।

```
int num = 12345;
String snum = String.valueOf(num);

double num = 123.45;
String snum = String.valueOf(num);
```

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 জাতার 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);
    }
}
```

```
}
}
```

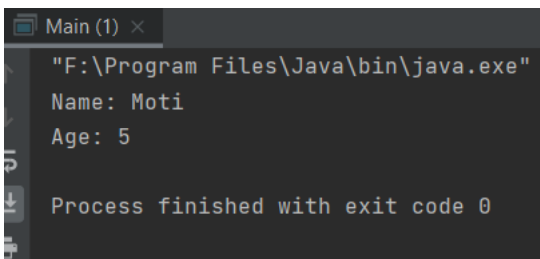
Animal ক্লাস এ একটি মেথড তৈরি করলাম PrintInformation নামে। এখন Dog class এর অবজেক্ট এর মাধ্যমে access করা যায় কিনা দেখি।

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।

অনুশীলনী

১। ইনহেরিটেন্স ব্যবহার করে নিচের ক্লাস গুলো তৈরি করতে হবে।

Class A	Class B
<pre> Int a,b; Int sum(int x,int y) { return x+y; } </pre>	<pre> Int c,d; </pre>

এখন নিচের কোডটি যেন কাজ করে তার জন্য ইনহেরিটেন্স তৈরি করতে হবে।

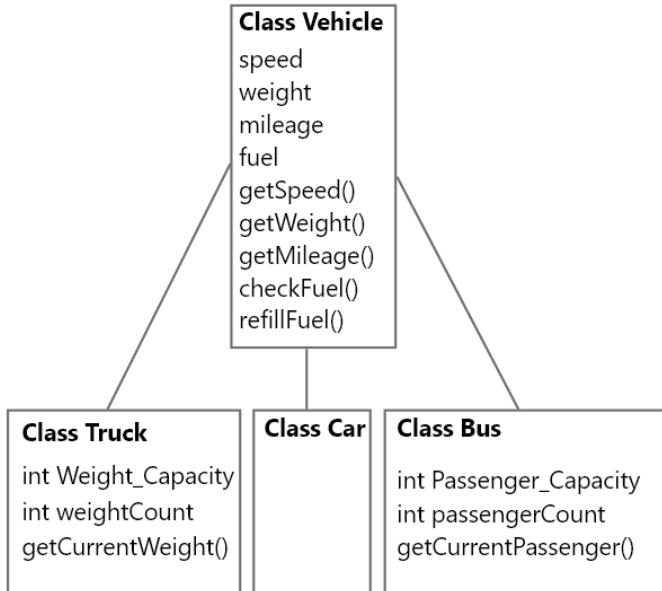
```
B b = new B();
```

```
b.c=100;
```

```
b.d=20;
```

```
int sum = b.sum(c,d);
```

২। নিচের ইনহেরিটেন্সটি তৈরি করতে হবে।



অ্যাক্সেস মডিফায়ারস ও এনকেপ্সুলেশন

এনকেপ্সুলেশন মানে হল এক ক্লাসের প্রোপার্টি অন্য ক্লাস থেকে অ্যাক্সেস নিয়ন্ত্রণ করা। জাভায় চারটি কীওয়ার্ড আছে যার সাহায্যে আমরা এনকেপ্সুলেশন করে থাকি। এদের অ্যাক্সেস মডিফায়ার বলে।

Modifier	Class	Package	Subclass	World
public	Yes	Yes	Yes	Yes
protected	Yes	Yes	Yes	No
No modifier	Yes	Yes	No	No
private	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।

অনুশীলনী

১। নিচের শর্তসমূহ মেনে একটি ক্লাস তৈরি করতে হবে।

Variables / Methods	Access scope
Int a Counter()	world
Int b; increaseB()	With in same class
Int c decreaseC()	package
Int d printSome()	With in package but if outside then from sub-class

গেটার এবং সেটার মেথড

গেটার এবং সেটার ব্যবহার করা হয় যেই প্রোপার্টি গুলো অ্যাক্সেস করা যায়না তাদের অ্যাক্সেস করার জন্য। যেমন প্রাইভেট প্রোপার্টি কে অন্য কোন ক্লাস থেকে অ্যাক্সেস করার জন্য। 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();
    }
}
```

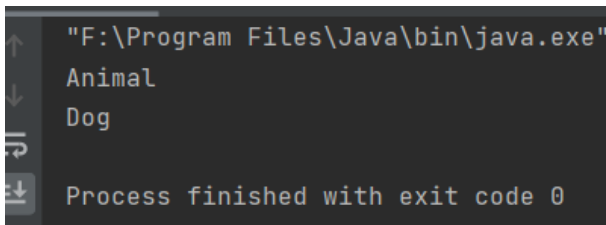
Dog ক্লাসের অবজেক্ট এর জন্য PrintInfo মেথডটি Dog প্রিন্ট করবে। কারণ Dog ক্লাসে PrintInfo মেথডকে অভাররাইড করে তার বডি পরিবর্তন করে দিয়েছি। অর্থাৎ System.out.println("Animal"); এর জায়গায় System.out.println("Dog"); এটি দিয়েছি।

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 ওভারলোডিং 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() // চর
{
    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 নম্বরের 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 নম্বরের 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 নম্বরের 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;
    }

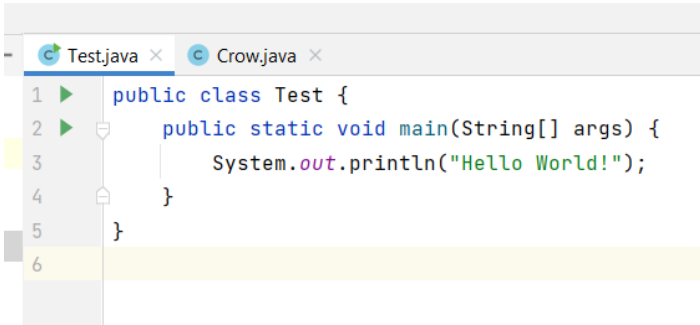
    static void sum()
    {
        System.out.println("No arguments");
    }

    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

আমরা মেইন মেথড এ সবসময় আর্গুমেন্ট হিসেবে স্ট্রিং এর Array দেখি। এটির কাজ আমরা অনেকেই জানিনা। এখন দেখব এর কি কাজ।



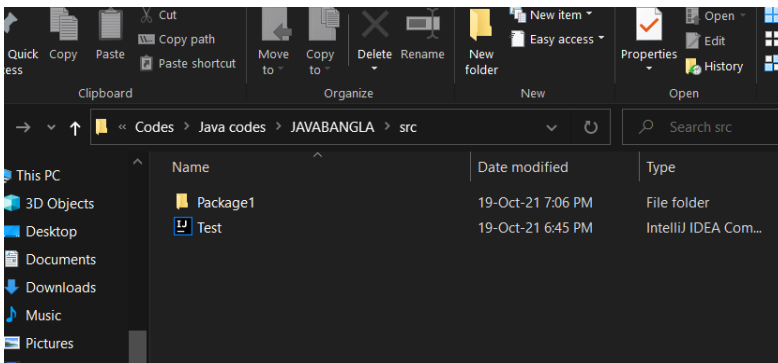
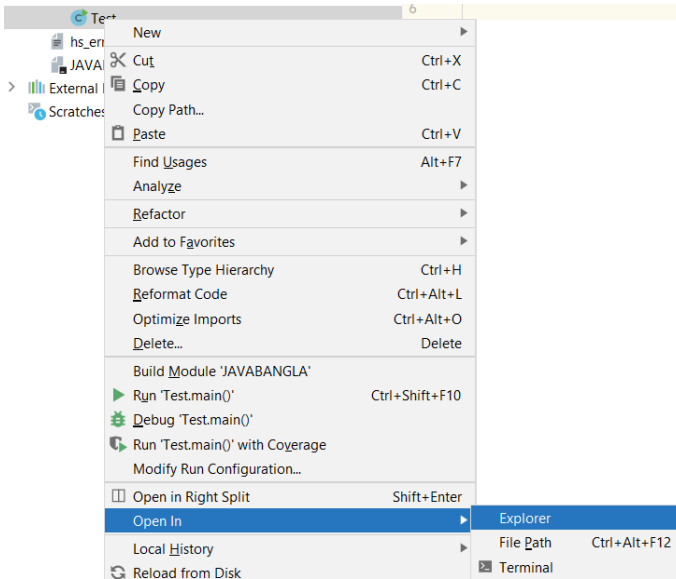
```

1 public class Test {
2     public static void main(String[] args) {
3         System.out.println("Hello World!");
4     }
5 }
6

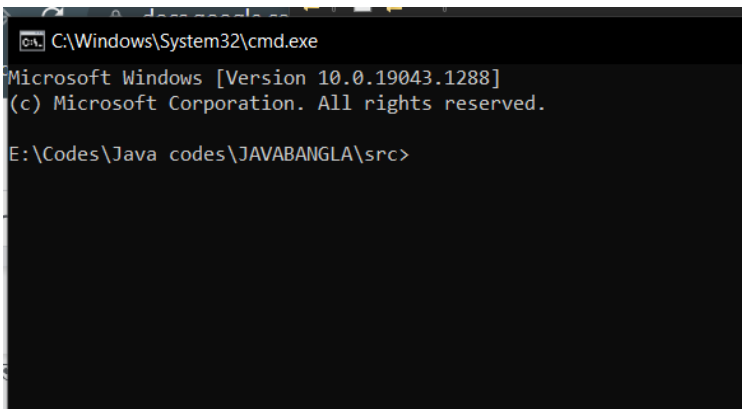
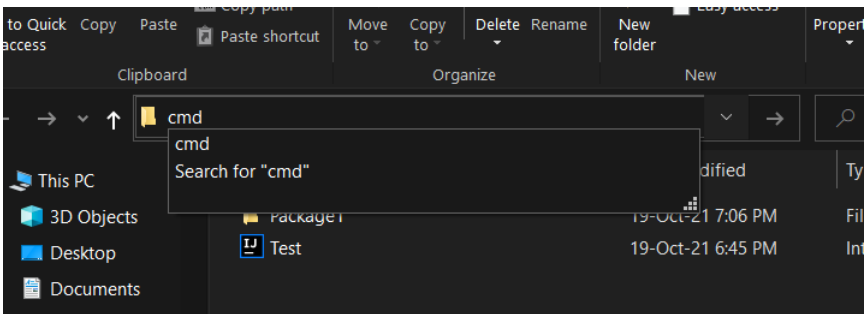
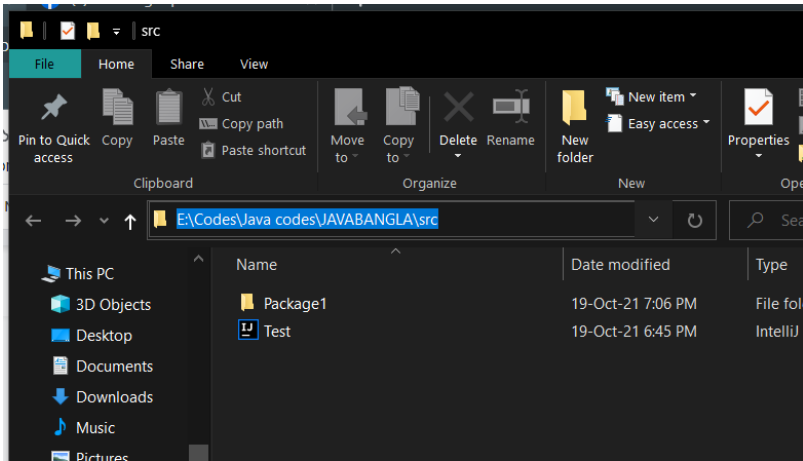
```

এটিই আসলে কমান্ড লাইন আর্গুমেন্ট। এটি দেখার জন্য আমাদের IDE ছেড়ে Command Prompt (CMD) এ যেতে হবে। এক্ষেত্রে জাভা ফাইলটি কোন প্রজেক্ট এর ভিতরে না রাখি। কারণ প্রজেক্টের ভিতরে থাকা জাভা ফাইল রান করার কমান্ড কিছুটা আলাদা। সহজভাবে দেখানোর জন্য প্রজেক্ট নিবো না এখন।

আমাদের জাভা প্রোগ্রামটি যে ফোল্ডার এ আছে অর্থাৎ src ফোল্ডারের লোকেশনে যাই। এর জন্য IntelliJ Idea তে প্রজেক্ট এর src ফোল্ডারের উপর মাউসের রাইট ক্লিক করে Open in > Explorer দিলে File Explorer দিয়ে ওপেন হয়ে যাবে।



এরপর File path এর উপর ক্লিক করলে সেখানে cmd টাইপ করতে হবে। এন্টার দিলেই CMD তে এই পথ ওপেন হয়ে যাবে।



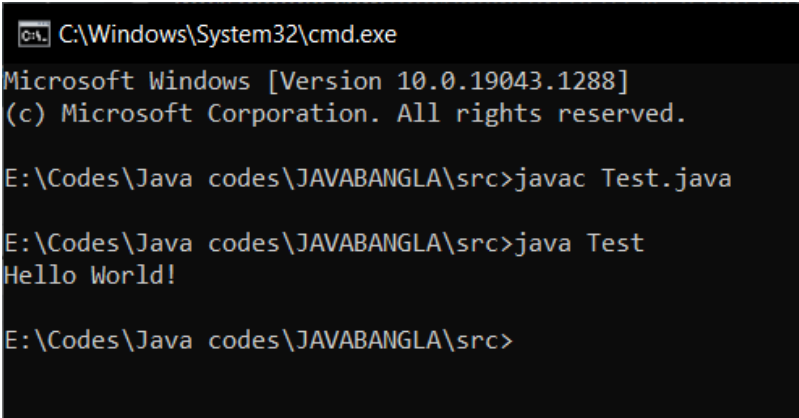
জাভা ফাইলটি cmd তে রান করানোর জন্য প্রথমে javac Test.java লিখে এন্টার দেই।

```
>javac Test.java
```

এখানে Test হচ্ছে জাভা ফাইলের নাম। এরপর java Test লিখে এন্টার দিলেই জাভা ফাইলটি রান হবে।

```
>java Test
```

এই কোডে আমরা Hello World! আউটপুট দেখতে পারবো।



```
C:\Windows\System32\cmd.exe
Microsoft Windows [Version 10.0.19043.1288]
(c) Microsoft Corporation. All rights reserved.

E:\Codes\Java codes\JAVABANGLA\src>javac Test.java

E:\Codes\Java codes\JAVABANGLA\src>java Test
Hello World!

E:\Codes\Java codes\JAVABANGLA\src>
```

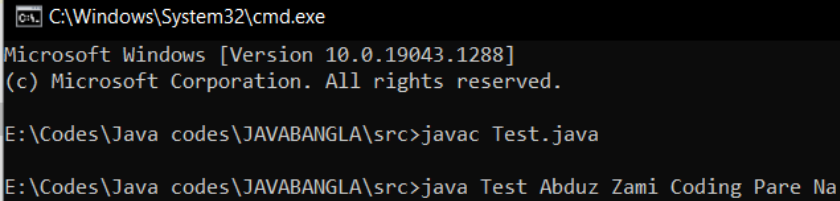
এতক্ষণ দেখলাম কিভাবে একটি জাভা ফাইল cmd তে রান করা যায়। এবার দেখব কমান্ড লাইন আর্গুমেন্ট। কমান্ড লাইন আর্গুমেন্ট রিখিত করা হয় String[] args এর মাধ্যমে। args একটি স্ট্রিং এর array। এই স্ট্রিং এর array প্রিন্ট করার জন্য আমরা কোডে কিছু পরিবর্তন আনি।

Test.java

```
public class Test {
    public static void main(String[] args) {
        for (String str:args
            ){
            System.out.println(str);
```

```
};
}
}
```

এখানে একটি for-each লুপ নিয়েছি args এর স্ট্রিং গুলো প্রিন্ট করার জন্য। এর পরের কাজ হচ্ছে আগের বারের মতই cmd তে চলে যাওয়া। এর পর আবার টাইপ করতে হবে javac Test.java এবং এন্টার প্রেস করতে হবে। এর পরের লাইনটিতে কিছু পরিবর্তন আছে। আগের বারের মতই java Test লিখতে হবে কিন্তু সাথে আমরা ইচ্ছা মত অনেক কিছু লিখতে পারি। এই ইচ্ছা মত লিখা জিনিসগুণগুলোই হল কমান্ড লাইন আর্গুমেন্ট। আর এই ইচ্ছা মত লিখা স্ট্রিং গুলোই args এর মাধ্যমে রিছিভ হয়।

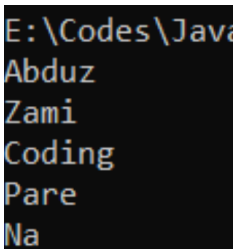


```
C:\Windows\System32\cmd.exe
Microsoft Windows [Version 10.0.19043.1288]
(c) Microsoft Corporation. All rights reserved.

E:\Codes\Java codes\JAVABANGLA\src>javac Test.java

E:\Codes\Java codes\JAVABANGLA\src>java Test Abdur Zami Coding Pare Na
```

এন্টার প্রেস করলে আমরা ওই ইচ্ছা মত লিখা স্ট্রিং বা কমান্ড লাইন আর্গুমেন্ট গুলো আউটপুটে দেখতে পারব।



```
E:\Codes\Java
Abdur
Zami
Coding
Pare
Na
```

this কি ওয়ার্ড

this কি ওয়ার্ড দিয়ে সেই ক্লাসের প্রোপার্টি নির্দেশ করে। উদাহরণ ছাড়া বোঝা কষ্টকর। তাই আমরা উদাহরণ দিয়ে বোঝার চেষ্টা করি।

আমরা প্রথম থগে শিখেছি কিভাবে একটি ক্লাসের কন্সট্রাক্টর তৈরি করতে হয়। তাহলে, আমরা একটি ক্লাস তৈরি করে তার একটি কন্সট্রাক্টর তৈরি করি।

```
public class Cats {
    int x,y,z;
    public Cats() {
    }

    public Cats(int a, int b, int c) {
        x = a;
        y = b;
        z = c;
    }
}
```

এখানে কন্সট্রাক্টর দিয়ে a b c এর মান নিয়ে তা x y z এ বসানো হয়েছে। এভাবে করতে কোন সমস্যা নেই। তবে যদি parameter এ x y z ব্যবহার করতে পারতাম তাহলে বেশ ভালই হত। অর্থাৎ

```
public Cats(int x, int y, int z) {
    x = x;
    y = y;
    z = z;
}
```

কিন্তু এ্ষেত্রে সমস্যা definition এ। x = x এ কম্পাইলার বুঝবে কিভাবে যে ঐ x টা কার। লোকাল variable নাকি ক্লাস এর instance variable।

এই সমস্যা দূর করতে this কি ওয়ার্ড ব্যবহার করা হয়।

কিভাবে করতে হয় সেটা দেখি এবার।

```
public class Cats {
    int x,y,z;
    public Cats() {
    }
    public Cats(int x, int y, int z) {
        this.x = x;
```

```
    this.y = y;  
    this.z = z;  
  }  
}
```

এখানে `this.x` দিয়ে বোঝানো হচ্ছে `Cats` ক্লাসের `instance`। আর `x` দিয়ে বোঝানো হচ্ছে লোকাল `variable` অথবা মেথডের `parameter` বা আর্গুমেন্ট এর `x`।

Super কীওয়ার্ড

Super কীওয়ার্ড ব্যবহার করা হয় তার ইমিডিয়েট প্যারেন্ট ক্লাসকে নির্দেশ করার জন্য। তিন টা কাজে Super কীওয়ার্ড ব্যবহার করা হয়। এগুলো হচ্ছে:

- ১। ইমিডিয়েট প্যারেন্ট ক্লাসের ভেরিএবলকে কল করার জন্য।
 - ২। ইমিডিয়েট প্যারেন্ট ক্লাসের মেথডকে কল করার জন্য।
 - ৩। ইমিডিয়েট প্যারেন্ট ক্লাসের কন্সট্রাক্টরকে কল করার জন্য।
- উপরের তিনটা কাজ মেথড ওভাররাইডিং এর সাথে জড়িত।

এবার আমরা কোডের মাধ্যমে Super keyword বোঝার চেষ্টা করি।

```
class X{
    //variable
    int x;

    //method
    void printSome(){
        System.out.println("Value of x: "+x);
    }

    //constructor
    public X(int x) {
        this.x = x;
    }
}
```

এই ক্লাসটিতে একটি ভেরিএবল একটি মেথড ও একটি কন্সট্রাক্টর ডিক্লেয়ার করা হয়েছে।

```
class Y extends X{
    int y;

    //constructor of class Y
    public Y(int x, int y) {
        super(x); // calling constructor of immediate parent class
        this.y = y;
    }

    //overriding the method
    @Override
    void printSome() {
        super.printSome(); // calling method of immediate parent class
    }
}
```

```

        System.out.println("Value of x: "+super.x+" Value of y: "+y); //calling variable of
        immediate parent class
    }
}

```

Y ক্লাসে যে কন্সট্রাক্টর ডিক্লেয়ার করা হয়েছে তার ভিতরে super(x) দিয়ে তার প্যারেন্ট ক্লাসের কন্সট্রাক্টরকে কল করা হয়েছে। যার মাধ্যমে x এর value initialize করা হয়েছে।

তার নিচে printSome() মেথডকে override করা হয়েছে। এখানে Super কীওয়ার্ড এর সাহায্যে variable কল করা এবং method কল করা দুইটিই দেখানো হয়েছে।

super.printSome() এর সাহায্যে মাতৃ ক্লাসের printSome() মেথডকে কল করা হয়েছে। এবং এর পরের লাইনে super.x এর মাধ্যমে মাতৃ ক্লাসের variable কে কল করা হয়েছে।

Static Variable

Static variable এমন এক ধরনের variable যা একটি ক্লাসের সব অবজেক্ট এর জন্য shared অবস্থায় থাকে। অর্থাৎ static variable এর মান পরিবর্তন করলে সব অবজেক্ট এর জন্যেই মান পরিবর্তিত হয়ে যাবে।

```
class Student{
    String student_name;
    String university_name;
}

public class bookstatic {
    public static void main(String[] args) {
        Student student1 = new Student();
        student1.student_name = "Abduz Zami";
        student1.university_name = "RUET";

        Student student2 = new Student();
        student2.student_name = "Shefat Zeon";
        student2.university_name = "RUET";
    }
}
```

এখানে Student ক্লাসের দুইটি অবজেক্ট তৈরি করা হয়েছে। দুইটি অবজেক্ট এর ই university_name একই। কিন্তু তাও বার বার ডিক্লেয়ার করতে হচ্ছে। এরকম যদি করা যেত যে সব অবজেক্ট এর জন্য university_name একই হবে, তাহলে বার বার ডিক্লেয়ার করতে হত না। এই কাজটি আমরা করতে পারি static variable এর মাধ্যমে।

```
class Student{
    String student_name;
    static String university_name;
}

public class booksuper {
    public static void main(String[] args) {
        Student.university_name = "RUET";

        Student student1 = new Student();
        student1.student_name = "Abduz Zami";
```

```
Student student2 = new Student();
student2.student_name = "Shefat Zeon";
```

```
System.out.println(Student.university_name);
}
}
```

এখানে `university_name` কে `static` করে দেওয়া হয়েছে। এখন আর প্রতিটি অবজেক্ট এর জন্য আলাদা আলাদা করে `university_name` বলে দেওয়ার প্রয়োজন নেই। সবার শুরুতে একবার বলে দিয়েছি। এতেই চলবে। `static variable` সকল অবজেক্ট সমান ভাবে `share` করে।

`static variable` নিয়ে কিছু গুরুত্বপূর্ণ পয়েন্ট:

১। `static variable` অ্যাক্সেস করার জন্য অবজেক্ট তৈরির প্রয়োজন নেই। `static variable` অ্যাক্সেস করার জন্য ক্লাসের নাম দিয়ে অ্যাক্সেস করতে হবে। অবজেক্ট এর নাম দিয়ে অ্যাক্সেস করা যাবে না। যেমন: উপরের কোডে `Student.university_name` এভাবে অ্যাক্সেস করা হয়েছে।

২। `static variable` প্রোগ্রাম শেষ না হওয়া পর্যন্ত `alive` থাকে। তাই লোকাল কোন ফাংশনে `static variable` ডিক্লেয়ার করলেও তা `global variable` এর মত আচরণ করে। `static variable` এর এই গুণের সাহায্যে আমরা কোন ক্লাসের অবজেক্ট এর সংখ্যা গণনা করতে পারি। এটা একটি কোডের সাহায্যে দেখব।

```
class Student{
    static int count = 0;
    String student_name;
    static String university_name;

    public Student(String student_name) {
        this.student_name = student_name;
        count++;
    }
}
```

```
public class booksuper {
    public static void main(String[] args) {
        Student.university_name = "RUET";

        Student student1 = new Student("Abduz Zami");
        Student student2 = new Student("Shefat Zeon");
        Student student3 = new Student("Naheen Kadir");

        System.out.println(Student.count);
    }
}
```


উপরের কোডে আমরা Student class এর অবজেক্ট গণনা করার জন্য একটি static টাইপের integer variable count নিয়েছি। কম্পট্রাক্টর এ count এর মান এক করে বাড়িয়েছি। এই count এর মাধ্যমেই আমরা জানতে পারবো আমাদের কতগুলো অবজেক্ট তৈরি করা হয়েছে।

Static variable এর value Student ক্লাসের প্রতিটি অবজেক্ট এর জন্য নতুন করে initialized হয় না। সে একবার ই initialized হয়। বোঝার জন্য এভাবে মনে রাখা যায়, static variable তার initialization এ লাইনে একবার মাত্র প্রবেশ করে এবং এর পর যতবার initialization লাইন আসবে সেই লাইনে আর ঢুকবে না।

static variable initialize না করলে 0 ধরে নিবে। যেমন উপরের কোডে count এর value 0 না দিলেও কাজ করত।

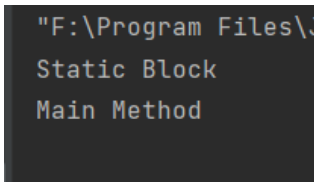
Static Block

Static block হচ্ছে কোডের এমন একটি অংশ যা মেইন মেথডের ও আগে execute হয়। আমরা static variable গুলো চাইলে static block এর ভিতরেও initialize করতে পারি। নিচে static block এর একটি কোড দেখি।

```
public class StaticBlock {
    static{
        System.out.println("Static Block");
    }

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

এই কোডের আউটপুট হবে:



```
"F:\Program Files\J
Static Block
Main Method
```

দেখা যাচ্ছে Static Block মেইন মেথডের আগে execute হয়েছে। অর্থাৎ, আমাদের মেইন মেথডের আগে কোন কাজ করার প্রয়োজন হলে আমরা টা static block এর ভিতরে করব।

Static variable গুলোকে আমরা চাইলে static block এর ভিতরে initialize করতে পারতাম।

```
class Student{
    static int count ;
    String student_name;
    static String university_name;

    static {
        count = 0;
    }
    public Student(String student_name) {
        this.student_name = student_name;
        count++;
    }
}
```

```
}
```

Static Method

Static variable এর মত static method কেও অবজেক্ট তৈরি না করে অ্যাক্সেস করা যায়। যেমনঃ

```
class EX{
    static int sum(int a, int b){
        return a+b;
    }
}

public class StaticMethod {
    public static void main(String[] args) {
        int s = EX.sum(5,6);
        System.out.println(s);
    }
}
```

Static method সম্পর্কে আরেকটি গুরুত্বপূর্ণ বিষয় হচ্ছে static method কে শুধু মাত্র static method এর ভিতরে কল করা যায় (অবজেক্ট তৈরি ছাড়া)।

```
class EX{
    static int sum(int a, int b){
        return a+b;
    }

    static void showAns(int a, int b){
        int ans = sum(a,b);
        System.out.println(ans);
    }
}
```

এখানে sum মেথডকে showAns মেথডের ভিতরে কল করার জন্য আমাদের showAns মেথডকেও static ডিক্লেয়ার করতে হয়েছে। showAns মেথড থেকে static কীওয়ার্ড সরিয়ে দিলে ERROR দেখতে পাবো।

মনে রাখতে হবে - main method যেহেতু static method তাই মেইন মেথডের ভিতরে কোন মেথডকে কল করতে হলে ঐ মেথডকে অবশ্যই static করতে হবে।

Static ক্লাস

একটি ক্লাসের সাথে স্ট্যাটিক কীওয়ার্ড ব্যবহার করা যাবে না যদি না এটি একটি ইনার (inner) ক্লাস হয়। একটি স্ট্যাটিক ইনার ক্লাস হল একটি নেস্টেড ক্লাস যা বাইরের ক্লাসের একটি স্ট্যাটিক মেম্বার। এটি অন্যান্য স্ট্যাটিক মেম্বার ব্যবহার করে, বাইরের ক্লাস থেকে বাইরের ক্লাসের অবজেক্ট তৈরি না করেই অ্যাক্সেস করা যেতে পারে।

একটা উদাহরণ দেখলে বিষয়টি ভালো ভাবে বোঝা যাবে।

```
public class Outer {
    static class Nested_Class {
        public void my_method() {
            System.out.println("This is my nested class");
        }
    }
    public static void main(String args[]) {
        Outer.Nested_Class nested = new Outer.Nested_Class();
        nested.my_method();
    }
}
```

ফাইনাল ভেরিএবল

ফাইনাল ভেরিএবল এমন একটি ভেরিএবল যার value আর পরিবর্তন করা যায় না।

```
public class FinalTest {
    final int cnt = 0;

    void changeFinal(){
        //we can not do the line below
        cnt = 100;
    }
}
```

ফাইনাল ভেরিএবলকে ডিক্লেয়ারেশন এর সময় বা কন্সট্রাক্টর এর ভিতরে initialize করা যায়।
ফাইনাল ভেরিএবলকে ডিক্লেয়ারেশন এর সময় initialize না করলে তাকে blank final variable বলে।
blank final variable কে কন্সট্রাক্টরের ভিতরে initialize করতে হয়।

```
class FinalTest {
    final int cnt = 0; // initialized in declaration
    final int pts;

    public FinalTest() {
        this.pts = 0; //initialized in constructor
    }
}
```

ফাইনাল মেথড

ফাইনাল মেথডকে অভ্যন্তরীণ করা যায় না।

```
class FinalTest {
    final void printSome(){
        System.out.println("final");
    }
}

class ABC extends FinalTest{
    //নিচের কাজটি করতে পারবো না
    @Override
    void changeFinal() {
        super.printSome();
    }
}
```

উপরের কোডটি রান করলে error পাবো। changeFinal() মেথডটি কে অভ্যন্তরীণ করতে দিবেনা কারণ এটি ফাইনাল মেথড।

ফাইনাল ক্লাস

ফাইনাল ক্লাসকে inherit করা যায় না।

```
final class FinalTest {
```

```
}
```

//নিচের লাইনে error পাবো

```
class ABC extends FinalTest{
```

```
}
```

এইভাবে আমরা ফাইনাল ক্লাসকে কোন ক্লাস বা ইন্টারফেস এ ইনহেরিট করতে পারবো না।

Static Final Variable

Static Final Variable এ static এবং final দুইটির বৈশিষ্ট্য বিদ্যমান। অর্থাৎ static হওয়ার কারণে সব অবজেক্টে সমান ভাবে shared থাকবে। একই সাথে initialization এর পর আর তার value পরিবর্তন করা যাবেনা।

Static final variable এর ডিক্লেয়ারেশন এরকম।

```
class FinalTest {
    static final int x=0;
}
```

আমরা Student ক্লাসের UniversityName প্রোপার্টি static final করতে পারি। কারন একে আর পরিবর্তন এর প্রয়োজন নেই। আর সবার জন্য একই নাম প্রযোজ্য।

```
class StudentData{
    static final String UniversityName = "RUET";
    String name;
}

public class FinalTest {
    public static void main(String[] args) {
        System.out.println(StudentData.UniversityName);
    }
}
```

Static final variable ডিক্লেয়ারেশন এর সময় initialize না করলে তাকে static blank final variable বলে। static blank final variable কে static block এ initialize করতে হয়।

```
class StudentData{
    static final String UniversityName;
    String name;
    static {
        UniversityName = "RUET";
    }
}

public class FinalTest {
    public static void main(String[] args) {
        System.out.println(StudentData.UniversityName);
    }
}
```

```
}
}
```

StringBuilder

প্রথম থগে স্ট্রিং নিয়ে বিস্তারিত আলোচনা করা হয়েছিল। সেখানে স্ট্রিং এর একটি অসুবিধার কথা বলা হয়েছিল। সেটি হল, স্ট্রিং Immutable Data Type। অর্থাৎ স্ট্রিং এ একবার value সেট করলে তা আর পরিবর্তন করা যায়না। এই সমস্যার সমাধান হচ্ছে এই স্ট্রিং বিল্ডার। স্ট্রিং বিল্ডার mutable data type এবং এর সাহায্যে আমরা value যখন ইচ্ছা যেভাবে ইচ্ছা পরিবর্তন করতে পারি।

স্ট্রিং বিল্ডার এর একটি উদাহরণ দেখি।

```
public class TestFile {
    public static void main(String[] args) {
        StringBuilder sb = new StringBuilder();
        StringBuilder sb2 = new StringBuilder("aaa");

        sb.append("AAA");
        sb.append(sb2);
        System.out.println(sb);
    }
}
```

এর আউটপুট

AAAaaa

স্ট্রিং বিল্ডার এ append এর মত মেথড গুলো ব্যবহার করলে মেইন স্ট্রিং এ পরিবর্তন হয়ে যায় যেটা String এ হত না।

স্ট্রিং বিল্ডার এ ব্যবহারযোগ্য কিছু মেথড দেখি।

1. `StringBuilder append(X x)`: This method appends the string representation of the X type argument to the sequence.
2. `int capacity()`: This method returns the current capacity.
3. `char charAt(int index)`: This method returns the char value in this sequence at the specified index.
4. `StringBuilder delete(int start, int end)`: This method removes the characters in a substring of this sequence.
5. `StringBuilder deleteCharAt(int index)`: This method removes the char at the specified position in this sequence.

6. `void ensureCapacity(int minimumCapacity)`: This method ensures that the capacity is at least equal to the specified minimum.
7. `void getChars(int srcBegin, int srcEnd, char[] dst, int dstBegin)`: This method copies characters from this sequence into the destination character array `dst`.
8. `int indexOf()`: This method returns the index within this string of the first occurrence of the specified substring.
9. `StringBuilder insert(int offset, boolean b)`: This method inserts the string representation of the `boolean` argument into this sequence.
10. `StringBuilder insert()`: This method inserts the string representation of the `char` argument into this sequence.
11. `int lastIndexOf()`: This method returns the index within this string of the last occurrence of the specified substring.
12. `int length()`: This method returns the length (character count).
13. `StringBuilder replace(int start, int end, String str)`: This method replaces the characters in a substring of this sequence with characters in the specified `String`.
14. `StringBuilder reverse()`: This method causes this character sequence to be replaced by the reverse of the sequence.
15. `void setCharAt(int index, char ch)`: In this method, the character at the specified index is set to `ch`.
16. `void setLength(int newLength)`: This method sets the length of the character sequence.
17. `CharSequence subSequence(int start, int end)`: This method returns a new character sequence that is a subsequence of this sequence.
18. `String substring()`: This method returns a new `String` that contains a subsequence of characters currently contained in this character sequence.
19. `String toString()`: This method returns a string representing the data in this sequence.
20. `void trimToSize()`: This method attempts to reduce storage used for the character sequence.

StringBuffer

স্ট্রিং বাফার এবং স্ট্রিং বিল্ডার প্রায় একই রকম কাজ করে। স্ট্রিং বিল্ডার এর প্রায় সব মেথড ই স্ট্রিং বাফার এর ক্ষেত্রেও প্রযোজ্য। তাই এই চ্যাপ্টার খুব বেশি দীর্ঘ করব না। আগের অধ্যায়ে স্ট্রিং বিল্ডার এর যতগুলো মেথড দেখানো হয়েছে সব গুলো স্ট্রিং বাফার এর জন্য ব্যাবহার করে দেখার পরামর্শ থাকল।

স্ট্রিং বাফার এর একটি উদাহরণ দেখি।

```
public class TestFile {
    public static void main(String[] args) {
        StringBuffer sb = new StringBuffer();
        StringBuffer sb2 = new StringBuffer("aaa");

        sb.append("AAA");
        sb.append(sb2);
        System.out.println(sb);
    }
}
```

স্ট্রিং বাফার এবং স্ট্রিং বিল্ডার এর পার্থক্য

স্ট্রিং বাফার	স্ট্রিং বিল্ডার
StringBuffer is synchronized i.e. thread safe. অর্থাৎ দুইটি থ্রেড একসাথে স্ট্রিং বাফার এর মেথড কে কল করতে পারেনা।	StringBuilder is non-synchronized i.e. not thread safe. অর্থাৎ দুইটি থ্রেড একসাথে স্ট্রিং বিল্ডার এর মেথড কে কল করতে পারে।
StringBuffer StringBuilder থেকে কম efficient	StringBuilder StringBuffer থেকে বেশি efficient

ArrayList

প্রথম থগে আমরা array শিখেছি। ArrayList কে dynamic array বলতে পারি। Array পড়ার সময় দেখেছি Array এর একটি ফিক্সড সাইজ থাকে, যা পরে আর পরিবর্তন করা যায়না। ArrayList এর সাইজ নির্দিষ্ট নয় এবং যখন ইচ্ছা তখন পরিবর্তন করা যায়। অর্থাৎ dynamically এর সাইজ পরিবর্তিত হয়। এছাড়া, ArrayList এ কিছু মেথড আছে যা আমাদের ArrayList এর ডাটাগুলোর উপরে বিভিন্ন অপারেশন করতে সাহায্য করে। নিচে ArrayList এর কিছু উদাহরণ দেখি।

```
import java.util.ArrayList;

public class TestFile {
    public static void main(String[] args) {
        ArrayList<Integer> arrayList = new ArrayList<>();

        // Appending new elements at
        // the end of the list
        for (int i = 0; i < 10; i++) {
            arrayList.add(i);
        }

        // Printing elements
        System.out.println(arrayList);

        // Remove element at index 3
        arrayList.remove(3);

        // Displaying the ArrayList
        // after deletion
        System.out.println(arrayList);

        // Printing elements one by one
        for (int i = 0; i < arrayList.size(); i++) {
            System.out.print(arrayList.get(i) + " ");
        }

        // Printing elements
        // Using for each loop
        for (Integer integer : arrayList) {
            System.out.print(integer + " ");
        }
    }
}
```

```

    }
}

```

উপরের কোডটি অবশ্যই রান করে আউটপুট দেখতে হবে।

ArrayList নিয়ে কিছু কথা বলি। যেহেতু ArrayList একটি dynamic array এবং এটি তৈরি করার সময় আমাদের সাইজ নির্দিষ্ট করতে হবে না, তাই যখন আমরা dynamically item যুক্ত করি এবং delete করি তখন ArrayList এর আকার স্বয়ংক্রিয়ভাবে বৃদ্ধি পায়। যদিও প্রকৃতপক্ষে আসল লাইব্রেরি বাস্তবায়ন আরও জটিল হতে পারে। জাভার ArrayList C++ এর vector এর মত। যদিও জাভায় vetor নামে আরেকটি ডাটা টাইপ আছে যার কাজ এবং ব্যবহার ArrayList এর মতই।

খুবই সাধারণভাবে বুঝানোর চেষ্টা করি। যখন ArrayList টি পূর্ণ হয়ে যায় এবং যদি আমরা একটি আইটেম যুক্ত করার চেষ্টা করি, তখন হিপ মেমরিতে একটি বড় আকারের মেমরি তৈরি হয় (সাইজের দ্বিগুণ আকারের)। এরপর বর্তমান মেমরির উপাদানগুলিকে নতুন মেমরিতে কপি করে। তারপর নতুন আইটেম যোগ করা হয়। কারণ এখন বড় মেমরি পাওয়া যাচ্ছে। সবশেষে পুরানো স্মৃতি মুছে ফেলে।

ArrayList এর ডিক্লারেশন এরকম।

```
ArrayList< Type > arrayList = new ArrayList<>();
```

Type হচ্ছে ArrayList এর element গুলোর ডাটা টাইপ। Type অবশ্যই রেকারেন্স টাইপ হবে। অর্থাৎ, Integer, Double, Character, String অথবা user defined যেকোনো ক্লাস। কিন্তু, int, float, double, char এগুলো ArrayList এর Type হতে পারেনা।

ArrayList এ ব্যবহারযোগ্য কিছু মেথড নিচে দেওয়া হল। এই মেথড গুলো ArrayList এর যেকোনো অবজেক্ট এর জন্য ব্যবহার করা যাবে। উদাহরণসরূপঃ

```
ArrayList< Type > arrayList = new ArrayList<>();
arrayList.add(Value);
```

মেথড	বর্ণনা
add(int index, Object element)	এর সাহায্যে একটি element কে লিস্টে নির্দিষ্ট ইনডেক্সে যুক্ত করা যায়।

add(Object o)	এর সাহায্যে একটি element কে লিস্টে যুক্ত করা যায়।
addAll(Collection C)	একটি লিস্টকে আরেকটি লিস্টে যুক্ত করা যায়।
addAll(int index, Collection C)	একটি লিস্টকে আরেকটি লিস্টের একটি নির্দিষ্ট ইনডেক্সে যুক্ত করা যায়।
clear()	লিস্টকে খালি করা যায়।
clone()	এটি লিস্টের একটি shallow copy রিটার্ন করে।
contains(Object o)	ঐ element কে খুঁজে পেলে true রিটার্ন করে অন্যথায় false।
get(int index)	কোন ইনডেক্স এর element রিটার্ন করে।
indexOf(Object O)	কোন element এর ইনডেক্স রিটার্ন করে। যদি না পায় তাহলে -1 রিটার্ন করে।
isEmpty()	লিস্ট খালি কিনা তা রিটার্ন করে।

lastIndexOf(Object O)	কোন element এর শেষ ইনডেক্স রিটার্ন করে। যদি না পায় তাহলে -1 রিটার্ন করে।
remove(int index)	নির্দিষ্ট ইনডেক্স এর element মুছে ফেলে।
remove(Object o)	কোন একটি নির্দিষ্ট element মুছে ফেলে।
removeAll?(Collection c)	এখানে মূলত একটি লিস্ট পাস করতে হয়। যদি পাস করা লিস্ট মেইন লিস্টে থাকে তাহলে সেগুলো মুছে ফেলবে।
removeRange(int fromIndex, int toIndex)	একটি রেঞ্জের মধ্যের element মুছে।
set?(int index, E element)	এর সাহায্যে একটি element কে লিস্টে নির্দিষ্ট ইনডেক্সে replace করা যায়।
size?()	লিস্টটির element সংখ্যা রিটার্ন করে।
toArray()	লিস্ট কে array বানিয়ে সেই array রিটার্ন করে।

ArrayList iterate করা পদ্ধতি

১। For loop

২। For Each loop

এই দুটি পদ্ধতিই উপরের কোডে উদাহরণ হিসেবে দেখানো হয়েছে।

ArrayList sorting

এবার দেখি কিতাবে ArrayList কে sort করতে হয়।

```
import java.util.ArrayList;
import java.util.Collections;

public class Main {
    public static void main(String[] args) {
        ArrayList<Integer> list = new ArrayList<>();
        list.add(3);
        list.add(2);
        list.add(6);
        list.add(1);
        System.out.println("unsorted: "+list);
        Collections.sort(list);
        System.out.println("sorted: "+list);
    }
}
```

এখানে সর্টিং এর জন্য Collection.sort() ব্যবহার করা হয়েছে। এই মেথডটি ArrayList কে ascending order এ সাজিয়ে দিবে।

Customized sorting এর জন্য comparator ব্যবহার করা যায়। যেমন নিচের কোডটি descending order এ সাজানোর জন্য।

```
import java.util.ArrayList;
import java.util.Collections;
import java.util.Comparator;

public class Main {
    public static void main(String[] args) {
        ArrayList<Integer> list = new ArrayList<>();
        list.add(3);
        list.add(2);
        list.add(6);
        list.add(1);
        System.out.println("unsorted: "+list);
        Collections.sort(list, new Comparator<Integer>() {
```

```

        @Override
        public int compare(Integer o1, Integer o2) {
            return o2-o1;
        }
    });
    System.out.println("sorted: "+list);
}
}

```

compare() মেথডটি o1 আর o2 কে compare করে positive সংখ্যা return করলে কোন পরিবর্তন করবেনা কিন্তু negative পেলে o1 আর o2 swap হবে।

এই comparator এর সাহায্যে object এর arraylist ও সর্ট করা যায়।

```

import java.util.ArrayList;
import java.util.Collections;
import java.util.Comparator;

class Man{
    String name;
    int age;

    public Man(String name, int age) {
        this.name = name;
        this.age = age;
    }
}

public class Main {
    public static void main(String[] args) {
        ArrayList<Man> list = new ArrayList<>();
        list.add(new Man("dasdas",12));
        list.add(new Man("fdfdzfsd",32));
        list.add(new Man("fsdf",11));
        list.add(new Man("xcsdfs",34));
        list.add(new Man("cxzc",78));
        Collections.sort(list, new Comparator<Man>() {
            @Override
            public int compare(Man o1, Man o2) {
                return o2.age-o1.age;
            }
        })
    }
}

```

```
});  
for (Man x:list  
    ) {  
    System.out.println(x.name+" "+x.age);  
}  
}  
}
```

আউটপুট

```
cxzc 78  
xcdfs 34  
fdfdzfsd 32  
dasdas 12  
fsdf 11
```

HashMap

HashMap একটি ডাটা টাইপ যাতে key-value pair থাকে। Array তে যেমন আমরা index এর মাধ্যমে তার value পেয়ে থাকি। HashMap এ key এর মাধ্যমে ঐ key এর against এ রাখা value পেতে পারি।

```
import java.util.HashMap;

public class TestFile {
    public static void main(String[] args) {
        HashMap<String,String> hashMap = new HashMap<>();
        hashMap.put("king", "Ertugrul");
        hashMap.put("minister", "Artuk");

        System.out.println(hashMap.get("king"));
        System.out.println(hashMap.get("minister"));
    }
}
```

আউটপুট

```
Ertugrul
Artuk
```

Key-value pair টি String, String ছাড়াও Integer, String অথবা Integer, Integer অথবা যেকোনো ধরনের pair হতে পারে।

HashMap এ ব্যবহারযোগ্য কিছু মেথড নিচে দেওয়া হল। এই মেথড গুলো HashMap এর যেকোনো অবজেক্ট এর জন্য ব্যবহার করা যাবে। উদাহরণসরূপঃ

```
HashMap < Type > hashmap = new HashMap <>();
hashmap .put(key,Value);
```

মেথড	বর্ণনা
clear()	সম্পূর্ণ ম্যাপ খালি করে

clone()	ম্যাপ এর একটি shallow copy রিটার্ন করে
containsValue(Object value)	যদি কোন key তে ঐ value থাকে তাহলে true রিটার্ন করে
entrySet()	সবগুলি entries এর একটি Set view রিটার্ন করে যা ম্যাপকে iterate করতে ব্যবহার হয়। নিচে iteration করার পদ্ধতি দেখার সময় বিস্তারিত দেখব।
get(Object key)	কোন key এর জন্য সেই value রিটার্ন করে। যদি কোন key না পায় তাহলে null রিটার্ন করে।
isEmpty()	ম্যাপ খালি কিনা চেক করে
keySet()	Key এর set view রিটার্ন করে। এটিও iterate করতে সাহায্য করে। নিচে iteration করার পদ্ধতি দেখার সময় বিস্তারিত দেখব।
put(K key, V value)	ম্যাপে নতুন ডাটা ইন্সার্ট করে।
putAll(Map<? extends K, ? extends V> m)	একটি ম্যাপে আরেকটি ম্যাপকে ইন্সার্ট করে
remove(Object key)	কোন একটি key এর value মুছে দেয়

size()	মাপের element সংখ্যা রিটার্ন করে
values()	Value গুলোর collection view রিটার্ন করে। এটিও iteration এ ব্যবহার হয়। নিচে iteration করার পদ্ধতি দেখার সময় বিস্তারিত দেখব।
getOrDefault(Object key, V defaultValue)	কোন key এর জন্য value খুঁজতে গেলে যদি null পায় অর্থাৎ কিছু না পায় সেক্ষেত্রে by default কি রিটার্ন করবে সেটি এই মেথডের সাহায্যে বলে দেওয়া যায়।
putIfAbsent(K key, V value)	যদি নির্দিষ্ট key ইতিমধ্যে একটি মানের সাথে যুক্ত না থাকে (অথবা নাল থাকে) তাহলে প্রদত্ত মানের সাথে এটি সংযুক্ত করে এবং নাল ফেরত দেয়, অন্যথায় বর্তমান value প্রদান করে।
remove(Object key, Object value)	নির্দিষ্ট কী-র জন্য এন্ট্রিটি সরিয়ে দেয় শুধুমাত্র যদি এটি বর্তমানে নির্দিষ্ট মানের সাথে ম্যাপ করা থাকে।
replace(K key, V value)	নির্দিষ্ট কী এর জন্য এন্ট্রি প্রতিস্থাপন করে শুধুমাত্র যদি এটি বর্তমানে কিছু মান ম্যাপ করা হয়।
replace(K key, V oldValue, V newValue)	নির্দিষ্ট কী-এর জন্য এন্ট্রি প্রতিস্থাপন করে শুধুমাত্র যদি বর্তমানে নির্দিষ্ট মানের সাথে ম্যাপ করা হয়।

HashMap iteration এর কতগুলো উপায় নিচে দেওয়া হল।

1. HashMap EntrySet বরাবর Iterator এর সাহায্যে
2. HashMap KeySet বরাবর Iterator সাহায্যে
3. for-each loop এর সাহায্যে
4. Lambda Expressions এর সাহায্যে.

HashMap EntrySet বরাবর Iterator এর সাহায্যে

```
import java.util.HashMap;
import java.util.Map;

// Class for iterating HashMap using for loop
public class HOJO {

    // Main driver method
    public static void main(String[] args)
    {
        // Creating a HashMap
        Map<String, String> foodTable
            = new HashMap<String, String>();

        // Inserting elements to the adobe HashMap
        // Elements- Key value pairs using put() method
        foodTable.put("A", "Angular");
        foodTable.put("J", "Java");
        foodTable.put("P", "Python");
        foodTable.put("H", "Hibernate");

        // Iterating HashMap through for loop
        for (Map.Entry<String, String> set :
            foodTable.entrySet()) {

            // Printing all elements of a Map
            System.out.println(set.getKey() + " = " +
                               +
                               set.getValue());
        }
    }
}
```

HashMap KeySet বরাবর Iterator সাহায্যে

```
import java.util.HashMap;
import java.util.Map;
```

```

public class HOJO {

    // Main driver method
    public static void main(String[] args)
    {
        // Creating hash map
        Map<Character, String> charType
            = new HashMap<Character, String>();

        // Inserting data in the hash map.
        charType.put('J', "Java");
        charType.put('H', "Hibernate");
        charType.put('P', "Python");
        charType.put('A', "Angular");

        // Iterating HashMap through forEach and
        // Printing all. elements in a Map
        charType.forEach(
            (key, value)
                -> System.out.println(key + " = " +
value));
    }
}

```

for-each loop এর সাহায্যে

```

import java.util.HashMap;
import java.util.Iterator;
import java.util.Map;
import java.util.Map.Entry;

public class HOJO {

    // Main driver method
    public static void main(String[] arguments)
    {
        // Creating Hash map
        Map<Integer, String> intType

```



```

        = new HashMap<Integer, String>();

// Inserting data(Key-value pairs) in hashmap
intType.put(1, "First");
intType.put(2, "Second");
intType.put(3, "Third");
intType.put(4, "Fourth");

// Iterator
Iterator<Entry<Integer, String> > new_Iterator
    = intType.entrySet().iterator();

// Iterating every set of entry in the HashMap
while (new_Iterator.hasNext()) {
    Map.Entry<Integer, String> new_Map
        = (Map.Entry<Integer, String>)
            new_Iterator.next();

// Displaying HashMap
System.out.println(new_Map.getKey() + " = "
                    +
new_Map.getValue());
}
}
}

```

Lambda Expressions এর সাহায্যে

Lambda Expressions আমরা এখন পর্যন্ত শিখিনি। এখন দেখে রাখ তাহলে পরবর্তীতে বুঝে নিতে পারবে।

```

import java.util.HashMap;
import java.util.Map;

// Class
public class H0J0 {

// Main driver method
public static void main(String[] args)

```

```

{
    // Creating hash map
    Map<Character, String> charType
        = new HashMap<Character, String>();

    // Inserting elements(key-value pairs)
    // in the hash map ( Custom inputs)
    charType.put('A', "Apple");
    charType.put('B', "Basketball");
    charType.put('C', "Cat");
    charType.put('D', "Dog");

    // Iterating through forEach and
    // printing the elements
    charType.forEach(
        (key, value)
            -> System.out.println(key + " = " +
value));
    }
}

```

HashSet

HashSet এমন একটি কালেকশন যাতে শুধুমাত্র unique উপাদান থাকে। যেমনঃ {1, 2, 5, 8} একটি সেট। কারণ এখানে কোন duplicate উপাদান নেই। কিন্তু {1, 2, 1, 5, 8} এটি সেট নয়। কারণ এতে 1 দুইবার এসেছে।

HashSet ডিক্লেয়ার করার নিয়মঃ

```
HashSet<Type> variable_name = new HashSet<>();
```

নিচে HashSet বোঝার জন্য ছোট একটি কোড দেখি।

```
import java.util.HashSet;

public class TestFile {
    public static void main(String[] args) {
        HashSet<Integer> hashSet = new HashSet<>();
        hashSet.add(1);
        hashSet.add(1);
        hashSet.add(1);
        hashSet.add(2);
        hashSet.add(3);
        hashSet.add(2);
        hashSet.add(3);
        hashSet.add(4);
        hashSet.add(5);

        System.out.println(hashSet);
    }
}
```

আউটপুটঃ

```
[1, 2, 3, 4, 5]
```

আউটপুট এ দেখা যাচ্ছে ইনপুট এ ডুপ্লিকেট ইলিমেন্ট থাকলেও আউটপুটে নাই।

HashSet এ ব্যবহারযোগ্য কিছু মেথড নিচে দেওয়া হল। এই মেথড গুলো HashSet এর যেকোনো অবজেক্ট এর জন্য ব্যবহার করা যাবে। উদাহরণস্বরূপ:

```
HashSet < Type > hashsap = new HashSet <>();
hashmap .insert(Value);
```

মেথড	বর্ণনা
add(E e)	এটি উপস্থিত না থাকলে নির্দিষ্ট element যোগ করতে ব্যবহৃত হয়, যদি এটি উপস্থিত থাকে তবে false রিটার্ন করে
clear()	সম্পূর্ণ সেট খালি করে দেয়
contains(Object o)	কোন element আছে কিনা চেক করে
remove(Object o)	কোন element রিমুভ করে
iterator()	সেটের একটি iterator রিটার্ন করে যা দিয়ে iteration করা যায়। এটি iteration এর পদ্ধতি তে দেখব আমরা।
isEmpty()	সেটটি খালি কিনা চেক করে।
size()	সেটের element সংখ্যা রিটার্ন করে।
clone()	সেটের একটি shalow copy রিটার্ন করে

এবার দেখি কিভাবে একটি HashSet ইটারেট করতে হয়।

পদ্ধতি ১: ইটারেটর এর সাহায্যে

```
import java.util.HashSet;
import java.util.Iterator;

public class TestFile {
    public static void main(String[] args) {
        HashSet<Integer> hashSet = new HashSet<>();

        hashSet.add(1);
        hashSet.add(1);
        hashSet.add(1);
        hashSet.add(2);
        hashSet.add(3);
        hashSet.add(2);
        hashSet.add(3);
        hashSet.add(4);
        hashSet.add(5);

        Iterator itr = hashSet.iterator();

        // Holds true till there is single element
        // remaining in Set
        while (itr.hasNext())
        {
            // Traversing elements and printing them
            System.out.print(itr.next() + " , ");
        }
        System.out.println();

    }
}
```

পদ্ধতি ২: For-Each লুপের সাহায্যে

```
import java.util.HashSet;

public class TestFile {
    public static void main(String[] args) {
        HashSet<Integer> hashSet = new HashSet<>();

        hashSet.add(1);
        hashSet.add(1);
        hashSet.add(1);
        hashSet.add(2);
        hashSet.add(3);
        hashSet.add(2);
        hashSet.add(3);
        hashSet.add(4);
        hashSet.add(5);

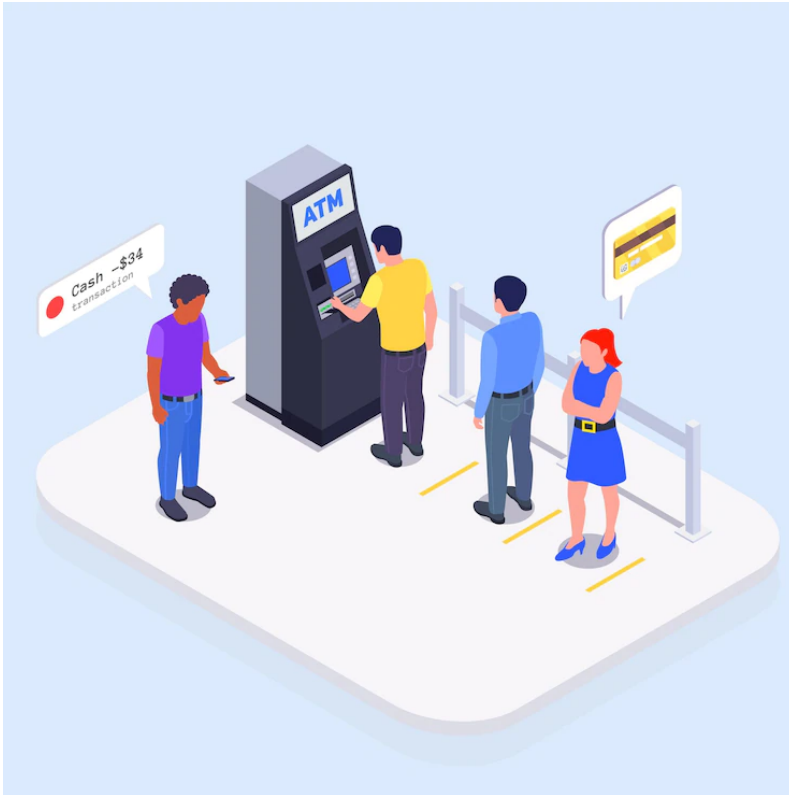
        // Using enhanced for loop for traversal
        for (Integer s : hashSet)

            // Traversing elements and printing them
            System.out.print(s + ", ");
        System.out.println();

    }
}
```

PriorityQueue

PriorityQueue বোঝার আগে Queue বুঝতে হবে। Queue এমন একটি data structure যা first-in-first-out নিয়ম মেনে চলে। Queue তে যে ডাটা আগে ঢুকে সেই বের করার সময় সেই ডাটা আগে বের হয়। জেমস: টিকিট কেনার জন্য লাইন। যে আগে দারাবে সেই আগে টিকেট কিনে লাইন থেকে বেরিয়ে যাবে।



PriorityQueue তে বাড়তি যে সুবিধা পাওয়া যায় তা হল এখানে ডাটাগুলো একটা priority মেনে সজ্জিত থাকে। Priority হতে পারে যে যত বড় সে আগে থাকবে বা যে ছোট সে আগে থাকবে এরকম। অর্থাৎ এখানে ডাটাকে sorted করে রাখা যায়। আর ডাটা বের করার সময় যে top এ থাকবে সে বের হবে।

PriorityQueue এর declaration।

```
PriorityQueue<Type> pq = new PriorityQueue<>();
```

PriorityQueue এর ছোট একটি কোড দেখি।

```
import java.util.PriorityQueue;

public class TestFile {
    public static void main(String[] args) {
        PriorityQueue<String> pq = new PriorityQueue<>();
        pq.add("Riyadh");
        pq.add("Mecca");
        pq.add("Medina");

        System.out.println(pq);
    }
}
```

PriorityQueue এর চারটি মেথড আমাদের জানতেই হবে।

- add(E element): এটি দিয়ে priority queue তে element insert করতে হয়।
- remove(): এটি সবার উপরের element কে remove করে।
- peek(): এটির সাহায্যে সবার উপরের element কে retrieve করা যায়। আর যদি Priority Queue খালি থাকে তবে null রিটার্ন করে।
- poll(): এটির সাহায্যে সবার উপরের element কে retrieve করা যায় এবং সাথে সাথে সেই element ডিলিট ও হয়ে যায়। আর যদি Priority Queue খালি থাকে তবে null রিটার্ন করে। PriorityQueue তে remove() এবং peek() অপারেশন একসাথে করা মানেই poll() করা।

remove() আর peek() অপারেশনের উদাহরণ নিচে দেওয়া হল।

```
import java.util.PriorityQueue;

public class TestFile {
    public static void main(String[] args) {
        PriorityQueue<String> pq = new PriorityQueue<>();
        pq.add("Riyadh");
        pq.add("Mecca");
        pq.add("Medina");

        System.out.println("Top element before removing :");
    }
}
```



```

"+pq.peek());
    pq.remove();
    System.out.println("Top element after removing :
"+pq.peek());

}
}

```

আউটপুট:

```

Top element before removing : Mecca
Top element after removing : Medina

```

poll() অপারেশন এর উদাহরণ নিচে দেওয়া হল।

```

import java.util.PriorityQueue;

public class TestFile {
    public static void main(String[] args) {
        PriorityQueue<String> pq = new PriorityQueue<>();
        pq.add("Riyadh");
        pq.add("Mecca");
        pq.add("Medina");

        System.out.println("Top element that is removed :
"+pq.poll());
        System.out.println("Current top element : "+pq.peek());

    }
}

```

আউটপুট:

```

Top element that is removed : Mecca
Current top element : Medina

```

উপরের কোডদুটি দেখলে remove(), peek() এবং poll() মেথড এর কাজ বোঝা যাবে আশা করি।

Priority Queue কে কিভাবে iterator এর সাহায্যে iterate করা যায় দেখি।

```

import java.util.Iterator;
import java.util.PriorityQueue;

public class TestFile {
    public static void main(String[] args) {
        PriorityQueue<String> pq = new PriorityQueue<>();
        pq.add("Riyadh");
        pq.add("Mecca");
        pq.add("Medina");

        Iterator iterator = pq.iterator();

        while (iterator.hasNext()) {
            System.out.print(iterator.next() + " ");
        }
        System.out.println();
    }
}

```

আমরা top element কে বার বার poll() করেও কাজটি করতে পারি। এক্ষেত্রে top element টি retrieve হতে থাকবে এবং ডিলিট হতে থাকবে। আর লুপ ততক্ষণ চলে যতক্ষণ না PriorityQueue খালি হচ্ছে।

```

import java.util.PriorityQueue;

public class TestFile {
    public static void main(String[] args) {
        PriorityQueue<String> pq = new PriorityQueue<>();
        pq.add("Riyadh");
        pq.add("Mecca");
        pq.add("Medina");

        while (!pq.isEmpty()) {
            System.out.print(pq.poll() + " ");
        }
        System.out.println();
    }
}

```


Anonymous Class

Anonymous class এমন একটি ক্লাস যে ক্লাসের কোন নাম থাকেনা। একটি সাধারণ ক্লাস তৈরি করি।

```
class Animal{
    void printAbout(){
        System.out.println("Animal");
    }
}
public class TestFile {
    public static void main(String[] args) {
        Animal dog = new Animal();
        dog.printAbout();
    }
}
```

এই কোডটি রান করলে অবশ্যই আউটপুট পাবো “Animal”। এই কোডে ছোট একটি পরিবর্তন করলেই আমরা anonymous class বানাতে পারবো।

```
class Animal{
    void printAbout(){
        System.out.println("Animal");
    }
}
public class TestFile {
    public static void main(String[] args) {
        Animal dog = new Animal(){
            @Override
            void printAbout() {
                System.out.println("Dog");
            }
        };
        dog.printAbout();
    }
}
```

এই কোডটি রান করলে আউটপুট আসবে “Dog”। এখানে **bold** করা অংশটিই হচ্ছে anonymous class। এখানে Animal class এর কিছু মেথড অভাররাইড করা হয়েছে। অনেকে বলতে পারে এটা তো Animal class। এটার তো নাম আছে। প্রকৃতপক্ষে এটার কোন নাম নেই। একটা ক্লাসের মেথডগুলো অভাররাইড

করতে হলে ঐ ক্লাসকে অন্য ক্লাস দ্বারা অভাররাইড করতে হয়। নতুন ক্লাসের একটি নাম ও দিতে হয়। এগুলো আমরা ইনহেরিটেন্স অধ্যায়ে দেখেছি। Anonymous class এ মেথড অভাররাইডিং এর জন্য কোন নতুন ক্লাস দিয়ে ইনহেরিট করা হয়নি। এখানে একটি নামহীন Anonymous class দিয়ে Animal class কে ইনহেরিট করা হয়েছে।

অনেক সময় প্রতিটি অবজেক্ট এর জন্য আলাদা আলাদা ধরনের মেথড লাগতে পারে। সেক্ষেত্রে এই Anonymous class উৎকৃষ্ট।

Anonymous class দিয়ে Abstract class ও interface কেও ইনহেরিট করা যায়।

Abstract Class

Abstract মানে হল ঝাপসা। এভাবে বলা যায় সব কিছু যেখানে বলা থাকেনা। Abstract Class এমন এক ধরনের ক্লাস যেখানে Abstract ও non-abstract দুই ধরনের মেথড ই থাকে। Abstract method এ কোন definition থাকেনা।

নিচে একটি Abstract class এর উদাহরণ দেওয়া হল।

```
abstract class Animal{
    void printWho(){
        System.out.println("Animal");
    }

    abstract void printDetails();
}
```

উপরের ক্লাসটিতে printWho() মেথডটি non-abstract। আর printDetails() মেথডটি abstract। printDetails() মেথডে কোন ডেফিনিশন নেই।

```
class Dog extends Animal{

    @Override
    void printDetails() {
        System.out.println("Dog");
    }
}
```

এই Dog ক্লাস Animal ক্লাসকে inherit করেছে। সেখানে Animal ক্লাসের abstract মেথড printDetails() কে অভাররাইড করা হয়েছে।

```
class Cat extends Animal{

    @Override
    void printDetails() {
        System.out.println("Cat");
    }
}
```

Cat ক্লাসটি দিয়েও একই কাজ করা হয়েছে।

```
public class TestFile {  
    public static void main(String[] args) {  
        Dog dog = new Dog();  
        dog.printDetails();  
  
        Cat cat = new Cat();  
        cat.printDetails();  
    }  
}
```

দেখা যাচ্ছে Animal ক্লাসকে Parent class ধরে আমরা তার abstract মেথড গুলো কে ইচ্ছা মত অভ্যন্তরীণ করে বিভিন্ন ক্লাসের জন্য ব্যবহার করতে পারছি।

Interface

Abstract Class এ Abstract ও non-abstract দুই ধরনের মেথডই ছিল। কিন্তু ইন্টারফেস এ সব মেথডই abstract হবে। তার সাথে ইন্টারফেস এ যত ভেরিএবল declare করা হবে সবই হবে public static final ভেরিএবল (by default)। By default এ কারনে বলা হচ্ছে যে এই ভেরিএবল গুলোকে declare করার সময় public static final লিখতে হয়না। এরার নিজে থেকেই public static final এর মত আচরণ করে।

যেমন: int x = 5; আর public static final int x = 5; ইন্টারফেস এর জন্য একই কাজ করে।

তেমনিভাবে, মেথড গুলোর খেত্রেও বলে দিতে হয়না এটি abstract method. কারন এখানে সব মেথডই by default abstract।

আরেকটি গুরুত্বপূর্ণ বিষয় হচ্ছে, ইন্টারফেস এ সব ভেরিএবল ও মেথড public। এখানে এদেরকে private বা protected করার সুযোগ নেই।

নিচে একটি ইন্টারফেস এর কোড দেওয়া হল।

```
public interface Dogs {
    int a = 10;
    void Display();
}
```

এখানে a ভিতরে ভিতরে public static final। static এর জন্য এর সকল অবজেক্ট এ একই মান থাকবে।

আর final এর জন্য এর মান পরিবর্তন করা যাবেনা।

আর Display() মেথডটিও public abstract ই হবে একই ভাবে।

ইন্টারফেস এর অবজেক্ট তৈরি করা যায়না। অবজেক্ট তৈরি করার জন্য অবশ্যই কোন একটি ক্লাস থেকে ইনহেরিট করে সেই ক্লাসের অবজেক্ট তৈরি করতে হবে। Anonymous class ও ব্যবহার করা যাবে।

এবার একটি ক্লাস থেকে উপরের ইন্টারফেসটি ইনহেরিট করি।

```
public class Tommy implements Dogs{
    @Override
    public void Display() {
        System.out.println("Tommy");
    }
}
```


এখানে সবকিছুই পরিচিত শুধুমাত্র implements কি ওয়ার্ডটি নতুন লাগছে। এটি নিয়ে পরের অধ্যায়েই বিস্তারিত বলব। আপাতত জেনে রাখলেই হবে যে interface কে যদি কোন class ইনহেরিট করে তাহলে implement ব্যবহার করতে হবে।

এখন আরেকটি ক্লাস থেকে Tommy এর অবজেক্ট তৈরি করতে পারি আমরা।

```
public class HojoBorolo{
    public static void main(String[] args) {
        Tommy t = new Tommy();
        t.Display();
    }
}
```

আউটপুট

Tommy

ইন্টারফেস মাল্টিপল ইনহেরিটেন্স সাপোর্ট করে। অর্থাৎ একটি ইন্টারফেস বা ক্লাস একাধিক ইন্টারফেস কে ইনহেরিট করতে পারি। একই ক্লাস কিন্তু একাধিক ক্লাসকে ইনহেরিট করতে পারেনা। যার জন্য জাভায়। ক্লাস মাল্টিপল ইনহেরিটেন্স সাপোর্ট করেনা। নিচে মাল্টিপল ইনহেরিটেন্স এর একটি উদাহরণ দেখি।

এবার আরও একটি ইন্টারফেস তৈরি করি।

```
public interface Elephant {
    void Display();
}
```

এখন একটি ক্লাস কে দুইটি ইন্টারফেস থেকেই ইনহেরিট করি।

```
public class Tommy implements Dogs, Elephant{
    @Override
    public void Display() {
        System.out.println("Tommy");
    }
}
```

আউটপুট

Tommy

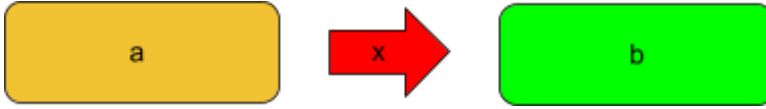
এবার একটি ইন্টারফেসে আমরা দুইটি ইন্টারফেস কে ইনহেরিট করি।

```
public interface Tommy extends Dogs, Elephant{
    void Show();
}
```

Tommy যেহেতু একটি ইন্টারফেস, সেহেতু Dogs আর Elephant এর মেথডগুলি আর অভাররাইড করার প্রয়োজন হয়নি। আর এখানে extend কি ওয়ার্ড ব্যবহার করা হয়েছে। কারন interface কে interface inherit করলে extend কি ওয়ার্ড ব্যবহার করতে হয়।

extends vs implements

নিচের ছবিটি লক্ষ্য করি।



ধরে নেই যে b বস্তুটি a কে inherit করছে। তাহলে x extends হবে নাকি implements হবে তা নিচের বিষয়গুলির উপর নির্ভর করবে।

Case 1	a b is of similar type like both of them are either class or interface.	x = extends
Case 2	a b is of different types like a is interface and b is class	x = implements

Lambda expression

এটি বেশ মজার একটি বিষয়। এটি কোড কে ছোট করে ফেলে একটি বিশেষ ক্ষেত্রে।

Interface এ যদি কেবল মাত্র একটি মেথড থাকে তাহলেই সেই মেথডকে আমরা ল্যাম্বডা এক্সপ্রেশন দিয়ে লিখতে পারি।

এবার একটি interface তৈরি করি। যার মাত্র একটি মেথড আছে।

```
public interface Dogs {
    void Display();
}
```

অন্য একটি ক্লাস থেকে anonymous class এর মাধ্যমে Dogs এর একটি অবজেক্ট তৈরি করি।

```
public class HojoBorolo{
    public static void main(String[] args) {
        Dogs dog = new Dogs() {
            @Override
            public void Display() {
                System.out.println("Dogs");
            }
        };
    }
}
```

উপরে যেভাবে অবজেক্ট তৈরি করেছি তা না করে এবার আমরা lambda expression ব্যবহার করে অবজেক্ট তৈরি করব। এখানে উপরের highlighted অংশ গুলোকে সরিয়ে ()-> দিয়ে replace করা হয়েছে।

```
public class HojoBorolo{
    public static void main(String[] args) {
        Dogs dog = () -> System.out.println("Dogs");
    }
}
```

এখানে Display() মেথডের ভিতরে মাত্র একটি লাইন থাকায় {} এর দরকার হয়নি। যদি একাধিক লাইন থাকতো তাহলে {} দিয়ে দিলেই হয়ে যেত। নিচে উদাহরণ দেখাচ্ছি।

```

public class HojoBorolo{
    public static void main(String[] args) {
        Dogs dog = () -> {
            String name = "Dogs";
            System.out.println(name);

        };
    }
}

```

আমরা চাইলে parameter ও পাস করতে পারি।

```

public interface Dogs {
    void Display(String name);
}

public class HojoBorolo{
    public static void main(String[] args) {
        Dogs dog = (String name) -> {
            System.out.println(name);

        };
    }
}

```

Exception handling

অনেক সময় প্রোগ্রামের কিছু জায়গায় কিছু exception/error এর জন্য প্রোগ্রাম ক্রাশ করে। Exception Handling এর মাধ্যমে সেইসব লাইনে exception/error আসলে কি করতে হবে বলে দেওয়া যায়। তাহলে প্রোগ্রামটি আর ক্রাশ করেনা।

ধরি নিচের লাইন গুলি একটি প্রোগ্রাম।

```
A
B
C
D
E
F
G
```

এখানে C D E এই তিনটি লাইনে error আসছে ধরে নেই। নরমালি প্রোগ্রাম এখানে ক্রাশ করবে। কিন্তু আমরা যদি এই তিন লাইনে exception handling করি তাহলে আর ক্রাশ করবেনা।

```
A
B
try{
C
D
E
}catch({})
F
G
```

এবার দেখি এই try/catch ব্লক কি জিনিস।

```
public class HojoBorolo{
    public static void main(String[] args) {
        int x = 5;
        int y = 0;
        System.out.println(x/y);
        System.out.println("Success");
    }
}
```

উপরের কোডটি রান করলে নিচের error টি পাওয়া যাবে।

Exception in thread "main" java.lang.ArithmeticException: / by zero
at HojoBorolo.main(HojoBorolo.java:5)

অর্থাৎ কোন সংখ্যাকে ০ দিয়ে ভাগ করা যায়না। তাই error দিচ্ছে।

```
public class HojoBorolo{
    public static void main(String[] args) {
        int x = 5;
        int y = 0;
        try{
            System.out.println(x/y);
        }catch (Exception e){
            System.out.println(e.getMessage());
        }
        System.out.println("Success");
    }
}
```

এবার সেই error সম্পন্ন লাইনটিতে আমরা try/catch ব্লক লাগিয়ে দেই। এরপর রান করি।

```
/ by zero
Success
```

এবার কিন্তু আর error দেখনি এবং প্রোগ্রাম ও শেষ পর্যন্ত রান করেছে। এখানে error ম্যাসেজটি আমরা প্রিন্ট করেছিলাম তাই সেটিও দেখাচ্ছে।

অর্থাৎ বোঝা গেল যে try এর ভিতরে প্রোগ্রাম এর যেই অংশে error দিতে পারে তাকে ঢুকিয়ে দিবো। আর catch ব্লকে error পেলে কি করতে হবে তা বলে দিবো। এখানে Exception দিয়ে সব ধরনের error কে নির্দেশ করে। আমরা চাইলে আলাদা আলাদা error এর জন্য আলাদা আলাদা instruction ও দিতে পারি। যেমন উপরের কোডের error/exception টি ছিল arithmetic exception। এখন আমরা arithmetic exception কেই শুধু handle করব।

```
import java.util.InputMismatchException;

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

```

int x = 5;
int y = 0;
try{
    System.out.println(x/y);
}catch (ArithmeticException e){
    System.out.println(e.getMessage());
}catch (IndexOutOfBoundsException e){
    System.out.println(e.toString());
}catch (InputMismatchException e){
    System.out.println(e.getMessage());
}finally {
    System.out.println("I will be printed always");
}
System.out.println("Success");
}
}

```

এখানে বিভিন্ন ধরনের exception এর জন্য বিভিন্ন ধরনের instruction দেওয়া হয়েছে। exception টি যে ধরনের হবে তার উপর ভিত্তি করে ঠিক করবে কোন catch ব্লকে যাবে। আর সবার শেষে finally নামে একটি ব্লক দেওয়া হয়েছে। এর কাজ হচ্ছে exception পাক না না পাক সে execute হবেই। এর আউটপুটটি দেখলেই বুঝতে পারবো।

```

/ by zero
I will be printed always
Success

```

আরেকটি বিষয় হচ্ছে যে এটি কোন ধরনের exception তা প্রিন্ট করার জন্য আমরা getMessage() বা toString() মেথড ব্যবহার করতে পারি।

আমরা একটি ফাংশনও ব্যবহার করতে পারতাম।

```

import java.util.InputMismatchException;

public class HojoBorolo{
    static double divide(int x, int y) throws ArithmeticException{
        return x/y;
    }
    public static void main(String[] args) {
        int x = 5;
        int y = 0;
        try{

```



```

        System.out.println(divide(x,y));
    }catch (ArithmeticException e){
        System.out.println(e.getMessage());
    }catch (IndexOutOfBoundsException e){
        System.out.println(e.getMessage());
    }catch (InputMismatchException e){
        System.out.println(e.getMessage());
    }finally {
        System.out.println("I will be printed always");
    }
    System.out.println("Success");
}
}

```

কোন ফাংশন যদি exception দেওয়ার সম্ভাবনা থাকে তাহলে আমরা ফাংশনের সাথে throws EXCEPTION দিয়ে দিতে পারি। যেমনটা উপরের কোডে divide() মেথডে করা হয়েছে। এটি কিন্তু কোন exception handle করে না।

আমরা নিজেদের ইচ্ছা মত exception ও throw করতে পারি।

```

static void testing(){
    throw new InputMismatchException();
}

```

আমরা নিজেদের জন্য কাস্টম exception তৈরিও করতে পারি। সেজন্য একটি ক্লাস খুলতে হবে যা Exception কে ইনহেরিট করবে।

```

class HabiJabi extends Exception{
    @Override
    public String getMessage() {
        return "HabiJabi Exception";
    }

    @Override
    public String toString() {
        return "HabiJabi Exception";
    }
}

```

এখন এটিকে আমরা যেকোনো জায়গা থেকে throw করতে পারি।

```
static void testing() throws Habijabi {  
    throw new Habijabi();  
}
```

ফাইল

ফাইলে read/write করার অনেকগুলো মেথড আছে। তার মধ্যে সবচেয়ে সহজ মেথডগুলোই এখানে দেখাবো।

ফাইলে read/write করার জন্য প্রথমে একটি File এর অবজেক্ট নিতে হবে।

```
File file = new File("az.txt");
```

Constructor এ যেই টেক্সট ফাইলে read/write করা হবে তার নাম দেওয়া হয়। এই ফাইলটি জাভা প্রোগ্রাম টি যেই ফোল্ডার এ আছে সেই ফোল্ডারে তৈরি হবে। অন্য এড্রেস এ তৈরি করতে চাইলে তার পথ দিয়ে দেওয়া যায়। যেমনঃ

```
File file = new File("E:\\Codes\\Java codes\\JAVA 12\\az.txt");
```

ফাইল থেকে read করার জন্য আমরা একটি Scanner এর অবজেক্ট নিব। Scanner এর constructor এ আগে তৈরি করার File এর অবজেক্টটি পাস করে দিবো। তাহলে এটি উপরোক্ত ফাইল থেকে সাধারণ Scanner দিয়ে read করার মত করেই read করা যাবে।

10

1 2 3 4 5 6 7 8 9 10

উপরের ফাইলটি নিচের কোডের সাহায্যে প্রিন্ট করা যাবে।

```
import java.io.File;
import java.io.FileNotFoundException;
import java.util.Scanner;

public class Main {

    public static void main(String[] args) throws
    FileNotFoundException {
        int t;

        File file = new File("az.txt");
        Scanner scanner = new Scanner(file);

        t = scanner.nextInt();
        int[] arr = new int[t];

        for (int i = 0; i < t; i++) {
            arr[i] = scanner.nextInt();
        }
    }
}
```

```

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

```

এবার দেখি কিভাবে ফাইলে write করা যায়। এর জন্য PrintWriter ব্যবহার করবো। উপরে যেই File এর অবজেক্ট তৈরি করেছিলাম তাকেই PrintWriter এর constructor এ পাস করে দিবো।

```
PrintWriter printWriter = new PrintWriter(file);
```

নিচের কোডটি execute করলে az.txt ফাইলে ০ থেকে ৯ পর্যন্ত সংখ্যা লিখবে।

```

import java.io.*;

public class Main {

    public static void main(String[] args) throws IOException {
        File file = new File("userdb.txt");
        PrintWriter printWriter = new PrintWriter(file);

        for (int i = 0; i < 10; i++) {
            printWriter.println(i);
        }
        printWriter.close();
    }
}

```

এখানে একটি প্রশ্ন জাগবেই। সেটি হচ্ছে printWriter.close(); এই লাইনটির মানে কি। এটি কমেন্ট আউট করে কোডটি রান করলে দেখা যাবে যে কিছুই write হয়নি। এর প্রয়োজন কি? printWriter কে close না করলে এটি write করেনা। আরও একটি মেথড আছে flush()। close আর flush এর পার্থক্য হচ্ছে close() কে কল করলে তা PrintWriter কে একেবারেই close করে দেয় এবং সেই PrintWriter এর অবজেক্ট দিয়ে আর write করা যায়না। flush() কে কল করলে তা কল করার আগ পর্যন্ত যা যা লিখতে বলা হয়েছে তা write করে দিয়ে আবারো write করার জন্য প্রস্তুত হবে। যখন আমাদের

write করার সাথে সাথেই read করার ও প্রয়োজন হবে তখন আমরা flush() ব্যবহার করবো। অর্থাৎ একবার write > read > write > read এরকম হলে।

এবার একই প্রোগ্রামে একসাথে read এবং write করা দেখব। তবে একটু কঠিনভাবে। কারন মাঝে মধ্যে নিজের comfort zone থেকে বের হয়ে আসতে হয় নাহলে সামনে আগানো যায়না।

```
Abduz Zami
10
fgdsfg Zami
100
Abduz Zdssgtsdmi
20
fsdfsd Zami
30
Abddsgdsuz
150
```

মনে করি আমাদের একটি ক্লাসের কতগুলো অবজেক্টকে ফাইলে সংরক্ষিত করতে হবে উপরের মত করে। সেই ক্লাসের দুইটি variable আছে। একটি name আরেকটি age। তারপর এটিকে read করতে হবে ফাইল থেকে।

```
import java.io.*;
import java.util.ArrayList;
import java.util.LinkedList;
import java.util.Scanner;

class User{
    String name;
    int age;

    public User() {
    }

    public User(String name, int age) {
        this.name = name;
        this.age = age;
    }

    public String getName() {
```

```

        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public int getAge() {
        return age;
    }

    public void setAge(int age) {
        this.age = age;
    }
}

public class Main {

    public static void main(String[] args) throws IOException {
        ArrayList<User> arrayList = new ArrayList<>();
        arrayList.add(new User("Abduz Zami",10));
        arrayList.add(new User("fgdsfg Zami",100));
        arrayList.add(new User("Abduz Zdssgtsdmi",20));
        arrayList.add(new User("fsdfsd Zami",30));
        arrayList.add(new User("Abddsgdsuz",150));

        File file = new File("userdb.txt");
        PrintWriter printWriter = new PrintWriter(file);

        for (User user:arrayList
        ) {
            printWriter.println(user.getName());
            printWriter.println(user.getAge());
        }
        printWriter.close();

        Scanner scanner = new Scanner(file);
        while (scanner.hasNextLine())
        {
            System.out.println("Name : "+scanner.nextLine());
        }
    }
}

```

```

        System.out.println("Age : "+scanner.nextLine());
    }
}

```

উপরের কোডটি রান করে দেখতে হবে। তাহলেই আশাকরি বোঝা যাবে কোডটি কিভাবে কাজ করছে।

এবার flush() এর কাজ দেখি।

```

import java.io.*;
import java.util.ArrayList;
import java.util.LinkedList;
import java.util.Scanner;

class User{
    String name;
    int age;

    public User() {
    }

    public User(String name, int age) {
        this.name = name;
        this.age = age;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public int getAge() {
        return age;
    }
}

```

```

    public void setAge(int age) {
        this.age = age;
    }
}

public class Main {

    public static void main(String[] args) throws IOException {
        ArrayList<User> arrayList = new ArrayList<>();
        arrayList.add(new User("Abduz Zami",10));
        arrayList.add(new User("fgdsfg Zami",100));
        arrayList.add(new User("Abduz Zdssgtsdmi",20));

        File file = new File("userdb.txt");
        PrintWriter printWriter = new PrintWriter(file);

        for (User user:arrayList
        ) {
            printWriter.println(user.getName());
            printWriter.println(user.getAge());
        }
        printWriter.flush();

        Scanner scanner = new Scanner(file);
        System.out.println("Print 1st time:");
        while (scanner.hasNextLine())
        {
            System.out.println("Name : "+scanner.nextLine());
            System.out.println("Age : "+scanner.nextLine());
        }

        arrayList.add(new User("fsdfsdsd Zami",30));
        arrayList.add(new User("Abddsgdsuz",150));

        for (User user:arrayList
        ) {
            printWriter.println(user.getName());
            printWriter.println(user.getAge());
        }
    }
}

```



```

        printWriter.flush();

        scanner = new Scanner(file);
        System.out.println("Print 2nd time:");
        while (scanner.hasNextLine())
        {
            System.out.println("Name : "+scanner.nextLine());
            System.out.println("Age : "+scanner.nextLine());
        }
    }
}

```

এখানে যেহেতু একবার write করে তারপর read করে তারপর আবার write করা হয়েছে। তাই এখানে flush() ব্যবহার করতে হয়েছে।

এখন পর্যন্ত আমরা যতবারই write করেছি ততবারই ফাইলে নতুন করে write হয়েছে এবং আগের জিনিস সব মুছে গিয়েছে। তাই এখন দেখব কিভাবে ফাইলে append করা যায়। অর্থাৎ আগের জিনিস আগের মতই থাকবে আর তার পরে নতুন জিনিস লিখবে।

এর জন্য আমাদের লাগবে FileWriter এর অবজেক্ট। File এর একটি অবজেক্টকে FileWriter এর constructor এ পাস করতে হবে।

```

import java.io.*;
import java.util.Scanner;

public class Main {

    public static void main(String[] args) throws IOException {
        // write your code here
        try{
            File file = new File("az.txt");
            FileWriter fileWriter = new FileWriter(file,true);
            int t = 5;
            while (t-->0) {
                fileWriter.write(t+" ");
            }
            fileWriter.close();
        }catch (Exception e){

```

```

        System.out.println(e.toString());
    }

}
}

```

উপরের কোডটি কয়েকবার রান করলে দেখা যাবে যতবার রান করা যাচ্ছে ততবার আগে যা লেখা হয়েছিল তার আগে যুক্ত হচ্ছে আর আগের লেখা থেকে যাচ্ছে।

এখানে চাইলে `PrintWriter` ও ব্যবহার করা যায়। নিচে একটি উদাহরণ দেওয়া হল।

```

import java.io.*;
import java.util.Scanner;

public class Main {

    public static void main(String[] args) throws IOException {
        // write your code here
        try{
            File file = new File("az.txt");
            FileWriter fileWriter = new FileWriter(file,true);
            PrintWriter printWriter = new PrintWriter(fileWriter);
            int t = 5;
            while (t-->0) {
                printWriter.println(t+" ");
            }
            fileWriter.close();
        }catch (Exception e){
            System.out.println(e.toString());
        }

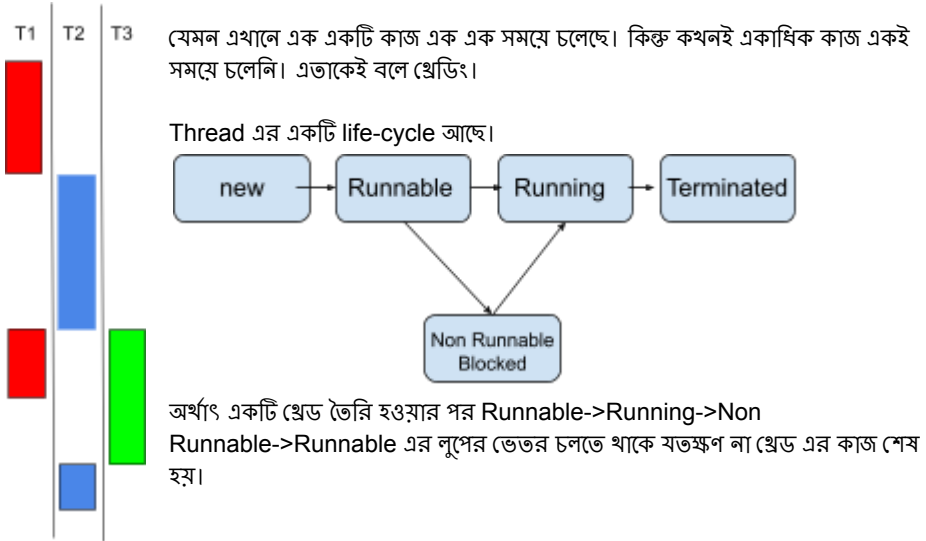
    }
}

```

এছাড়াও `FileInputStream` দিয়ে `read` আর `FileOutputStream` দিয়ে `write` করা যায়।

Multi-threading

কম্পিউটার এর সিপিইউ একসাথে একাধিক কাজ করতে পারেনা। একটি কাজ শেষ হওয়ার পর আরেকটি কাজ হয়। থ্রেড একই সাথে একাধিক কাজ করতে সাহায্য করে। একসাথে বলতে একটি কাজ কিছুক্ষন করে তার পর অন্য একটি কাজ করে। তার কিছুক্ষন পর আবার আরেকটি কাজ করে। তারপর আবার ধরে নেও যে আগের কাজে ফিরে যায় এভাবে চলতে থাকে।



থ্রেডিং কয়েকভাবে করা যায়। আমরা তিন ভাবে থ্রেডিং শিখব।

- ১। Extending Thread Class
- ২। Implementing Runnable Interface
- ৩। Creating thread by passing runnable object

Extending Thread Class

নিচে একটি ক্লাস ABC কে দিয়ে Thread কে ইনহেরিট করা হয়েছে। Thread এ একটি মেথড আছে run()। এই run() মেথডকে override করে যা লিখা হবে তাই thread টি running থাকলে run হবে।

```

class ABC extends Thread{
    private int a;
    private int b;
  
```

```

ABC(int j,int k){
    a = j;
    b = k;
}
public void run(){
    for(int i=b;i>0;i--){
        System.out.println("THREAD: "+a+" Prints: "+i);
    }
}
}

public class Main {
    public static void main(String[] args) {
        ABC obj1 = new ABC(1,3000000);
        ABC obj2 = new ABC(2,4000000);
        obj1.start();
        obj2.start();
        try{
            obj1.join();
            obj2.join();
        }catch (Exception e){
            System.out.println(e.getMessage());
        }
    }
}

```

এখানে x.join() মেথড যতক্ষণ x এর কাজ শেষ না হচ্ছে ততক্ষণ অন্যান্য থ্রেড গুলোকে terminate হতে বাঁধা দেয়।

Implementing Runnable Interface

নিচের কোডে একটি ক্লাস ABC কে দিয়ে Runnable interface কে ইমপ্লিমেন্ট করা হয়েছে। তারপর মেইন মেথডে Thread এর অবজেক্টে ABC এর অবজেক্ট পাঠানো হয়েছে।

```

class ABC implements Runnable{
    private int a;
    private int b;
    ABC(int j,int k){

```

```

        a = j;
        b = k;
    }
    public void run(){
        for(int i=b;i>0;i--){
            System.out.println("THREAD: "+a+" Prints: "+i);
        }
    }
}

public class Main {
    public static void main(String[] args) {
        ABC a_obj1 = new ABC(1,30);
        ABC a_obj2 = new ABC(2,40);
        Thread ob1 = new Thread(a_obj1) ;
        Thread ob2 = new Thread(a_obj2) ;
        ob1.start();
        ob2.start();
        try{
            obj1.join();
            obj2.join();
        }catch (Exception e){
            System.out.println(e.getMessage());
        }
    }
}

```

Creating thread by passing runnable object

এখানে আলাদা ভাবে ক্লাস তৈরি না করে anonymous class এর সাহায্যে Thread তৈরি করা হয়েছে।

```

public class Main {
    public static int count = 0;
    public static void increase(){
        count++;
        System.out.println(count);
    }
}

```

```

    }
    public static void decrease(){
        count--;
        System.out.println(count);
    }
    public static void main(String[] args) {
        Thread t1 = new Thread(new Runnable() {
            @Override
            public void run() {
                for (int i = 0; i < 100000; i++) {
                    increase();
                }
            }
        });

        Thread t2 = new Thread(new Runnable() {
            @Override
            public void run() {
                for (int i = 0; i < 100000; i++) {
                    decrease();
                }
            }
        });

        t1.start();
        t2.start();

        try{
            t1.join();
            t2.join();
        }catch (Exception e){
            System.out.println(e.getMessage());
        }

        System.out.println("final: "+count);
    }
}

```



এতক্ষণ এক এক মেথড এক এক সময় চলছিল। এখন আমরা যদি এরকম করতে চাই যে যতক্ষণ না একটি থ্রেড শেষ না হচ্ছে ততক্ষণ অন্য থ্রেড চলবেনা তাহলে কি করবো? উত্তর হচ্ছে ঐ মেথডটা কে `synchronized` করে দিবো। তাহলে একটি মেথড শেষ না হওয়া পর্যন্ত বাকি থ্রেড গুলো অপেক্ষা করবে।

নিচে একটি কোডের মাধ্যমে বোঝার চেষ্টা করি। কোডটি রান করে দেখতে হবে। তাহলেই আউটপুট থেকে বোঝা যাবে একটি থ্রেড সম্পূর্ণ হওয়ার পরই অন্য থ্রেড `execute` হচ্ছে।

```
public class Main {
    public static int count = 0;
    public synchronized static void increase(){
        count++;
        System.out.println(count);
    }
    public synchronized static void decrease(){
        count--;
        System.out.println(count);
    }
    public static void main(String[] args) {
        Thread t1 = new Thread(new Runnable() {
            @Override
            public void run() {
                for (int i = 0; i < 100000; i++) {
                    increase();
                }
            }
        });

        Thread t2 = new Thread(new Runnable() {
```

```

@Override
public void run() {
    for (int i = 0; i < 100000; i++) {
        decrease();
    }
}

});

t1.start();
t2.start();

try{
    t1.join();
    t2.join();
}catch (Exception e){
    System.out.println(e.getMessage());
}

System.out.println("final: "+count);
}
}

```

এর আউটপুট দেখলেই বোঝা যাবে যে আগে count বাড়তে বাড়তে সর্বোচ্চ হয়েছে তার পরে কমতে শুরু করেছে।

আরেকটি সহজ Threading এর উদাহরণ দিয়ে শেষ করবো।

আমরা চাইলে একটি নির্দিষ্ট সময়ের জন্য কোন টাস্ক কে থামিয়ে রাখতে পারি।

```

public class Main {
    public static void main(String[] args) {
        System.out.println("start");
        try{
            Thread.sleep(5000);
        }catch (Exception e){
            System.out.println(e.getMessage());
        }
        System.out.println("stop");
    }
}

```


এখানে start প্রিন্ট হওয়ার ৫ সেকেন্ড পর stop প্রিন্ট হচ্ছে।
সমাপ্ত