বাংলায় জাভা

howtocode.com.bd

Published
with GitBook

Table of Contents

howtocode.com.bd

Like 3,164

জাভা প্রোগ্রামিং

GITTER JOIN CHAT →


rokon12 118


nuhil 11


howtocode-com-bd 4


shabnam611 2

সংক্ষেপ

কোর্সের বর্ণনা: জাভা বর্তমানে বহুল ব্যবহৃত একটি প্রোগ্রামিং ল্যাংগুয়েজ। এন্টারপ্রাইজ এপ্লিক্যাশান ডেভেলেপমেন্টে এখনো জাভার বিকল্প তৈরি হয়নি বলে ধরা হয়। জাভার জনপ্রিয়তার মুল কারণ এর portability, নিরাপত্তা, এবং অবজেক্ট ওরিয়েন্টেড প্রোগ্রামিং ও ওয়েব প্রোগ্রামিং এর পরিপূর্ণ সাপোর্ট। এই কোর্সে জাভার অ আ ক খ থেকে শুরু করে এর ব্যবহারিক প্রয়োগ এবং অন্যান্য বিষয় গুলো নিয়ে আলোচনা করা হবে।

কাদের জন্যে কোর্স: এই কোর্স মূলত বিশ্ববিদ্যালয় এর প্রথম বর্ষের ছাত্র-ছাত্রীদের জন্যে যারা অবজেক্ট ওরিয়েন্টেড কনসেপ্ট শুরু করতে চায়। তবে যে কেও চাইলে এই কোর্সটি করতে পারে। ধরে নেওয়া হচ্ছে যে, শিক্ষার্থী অন্তত যে কোন একটি প্রোগ্রামিং ল্যাংগুয়েজ (সি/সি++) সম্পর্কে আগে থেকেই ধারণা রাখে।

> ## Statutory warning
>
> **This book may contain unexpected misspellings. Reader Feedback Requested.**

ওপেন সোর্স

এই বইটি মূলত স্বেচ্ছাশ্রমে লেখা এবং বইটি সম্পূর্ন ওপেন সোর্স । এখানে তাই আপনিও অবদান রাখতে পারেন লেখক হিসেবে । আপনার কন্ট্রিবিউশান গৃহীত হলে অবদানকারীদের তালিকায় আপনার নাম যোগ করে দেওয়া হবে ।

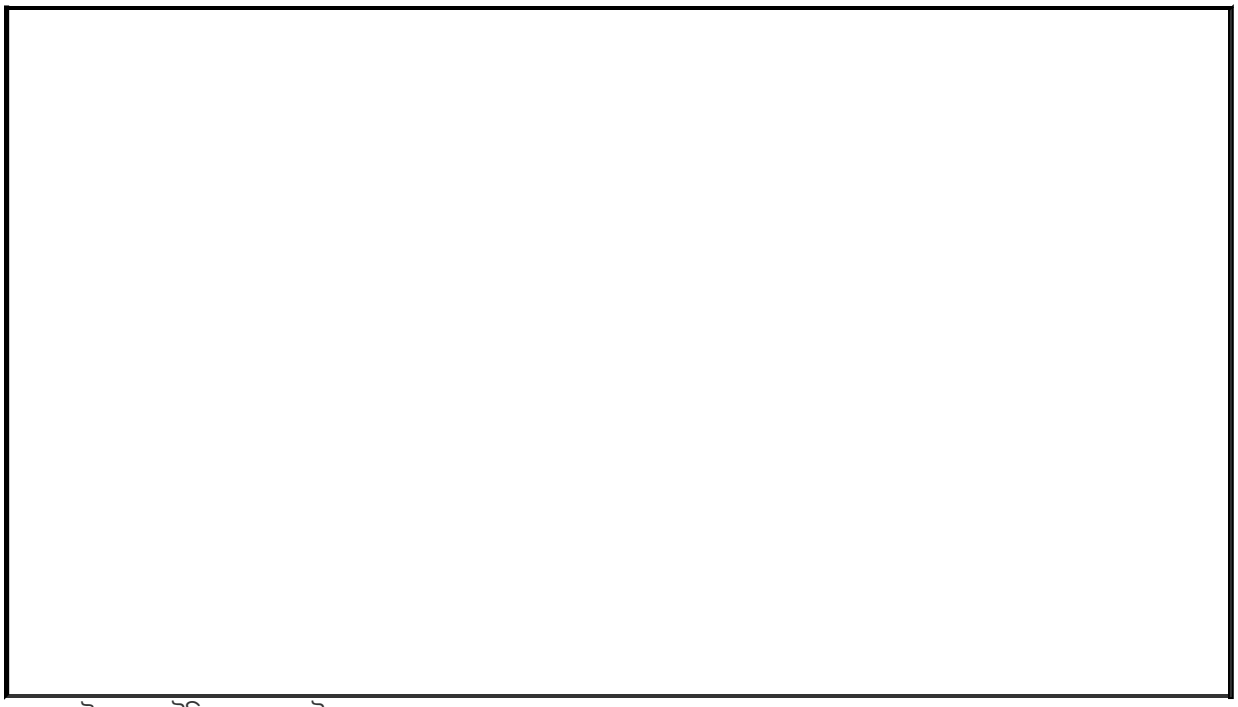
এটি মূলত একটি গিটহাব রিপোজিটোরি যেখানে এই বইয়ের আর্টিকেল গুলো মার্কডাউন ফরম্যাটে লেখা হচ্ছে । রিপোজটরিটি ফর্ক করে পুল রিকুয়েস্ট পাঠানোর মাধ্যমে আপনারাও অবদান রাখতে পারেন । বিস্তারিত দেখতে পারেন এই ভিডিওতে Video

উপক্রমণিকা

১৯৯৫ সালের ২৩ শে মে। ঝকঝকে ঝলমলে চমৎকার একটি দিন। জন গেইজ, ডিরেক্টর অব সান মাইক্রোসিস্টেম সাথে Marc Andreesen, কো ফাওন্ডার এবং ভাইস প্রেসিডেন্ট অব নেটস্কেপ ঘোষণা দেন যে, জাভা টেকনোলজি মোটেই কোন উপকথা নয়, বরং এটিই বাস্তবতা এবং তারা এটি Netscape Navigator এর সংযুক্ত হতে যাচ্ছে।

সে সময় জাভাতে কাজ করে এমন লোকের সংখ্যা ত্রিশেরও কম। তারা কখনোই চিন্তা করে নি, তাদের এই টিম ভবিষ্যৎ পৃথিবীর প্রধানতম টেকনোলজি নির্ধারণ করতে যাচ্ছে। ২০০৪ সালের ৩ জানুয়ারী Mars Exploration Rover মঙ্গল গ্রহের মানটিতে পা রাখে যার কন্ট্রোল সিস্টেম থেকে শুরু করে পৃথিবীর অধিকাংশ কনজুমার ইলেকট্রনিক্স - (ক্যাবল সেট-টব বক্স, ডিসিআর, টোস্টার, পিডিএ, স্মার্টফোন) ৯৭% এন্টাপ্রাইজ ডেস্কটপ ৮৯% ডেস্কটপ অব ইউএসএ, ৩ বিলিওন মোবাইল ফোন, ৫ বিলিওন জাভা কার্ড, ১২৫ মিলিওন টিভি ডিভাইস, ১০০% ব্লু-রে ডিস্ক প্লেয়ার … এই লিস্ট লম্বা হতেই থাকবে) জাভা রান করে।

নিচের ভিডিও টি চমৎকার। একবার দেখে নেওয়া যেতে পারে।



চলুন একটু পেছনের ইতিহাস জেনে নেই।

তখন সি-প্লাস প্লাস এর একচ্ছত্রাধিপত্য।

সান মাইক্রাসিস্টেম- মূলত হার্ডওয়্যার কম্পানি। ১৯৭২ থেকে ১৯৯১ সালে কম্পিউটারের হার্ডওয়্যারের এক রেভ্যুলেশান হয়। দ্রুত এবং উচ্চ ক্ষমতা সম্পন্ন হার্ডওয়্যার অল্প দামে পাওয়া যাচ্ছে এবং সেই সাথে কমপ্লেক্স সফটওয়্যারের চাহিদা দ্রুতই বেড়ে যাচ্ছে। ১৯৭২ Dennis Ritchie সি প্রোগ্রামিং ল্যাংগুয়েজ ডেভেলপ করেন যা প্রোগ্রামারদের মধ্যে সব থেকে জনপ্রিয়। কিন্তু ততদিনে প্রোগ্রামারদের কাছে সি -এর স্ট্রাকচার্ড প্রোগ্রামিং কিছুটা ক্লান্তিকর মনে হতে শুরু করেছে। এর ফলশ্রুতিতে Bjarne Stroustrup 1979 সালে ডেভেলপ করে সি প্লাস প্লাস যা কিনা সি এর এনহান্সমেন্ট । এটি সাথে অবজেক্ট ওরিয়েন্টেড প্রোগ্রামিং ধারণাকে পরিচিত করে তুলে। অবজেক্ট ওরিয়েন্টেড প্রোগ্রামিং এর সুবিধে হচ্ছে প্রোগ্রামার পুনর্ব্যবহারযোগ্য(reusable) কোড লিখতে পারে যা কিনা পরে অন্য কাজে পুনরায় ব্যবহার করা যায়।

১৯৯০ সাল। সান মাইক্রাসিস্টেম -এ সি প্লাস প্লাস এর আধিপত্যে সি-তে লেখা টুল এবং এপিআই গুলো প্রায় অবস্যুলেট হতে শুরু করেছে। Patrick Naughton,ইঞ্জিনিয়ার অব সান মাইক্রাসিস্টেম, মোটামুটি হতাশ এবং এক ধরণের অকওয়্যার্ড পরিস্থিতির স্বীকার। ততদিনে স্টিভ জব অ্যাপল কম্পিউটার থেকে বিতাড়িত হয়ে NeXT Computer, Inc প্রতিষ্ঠা করে ফেলেছেন(যা কিনা সফটওয়্যার ইন্ডাস্ট্রিতে বৈপ্লবিক পরিবর্তন আনতে যাচ্ছে এবং যার ফলশ্রুতিতে তৈরি হয়েছে আজকের ম্যাক-ওস) এবং NeXTSTEP নামে একটি অপারেটিং সিস্টেম তৈরি করেন। এতে কিছু অসাধারণ ব্যাপার ছিল যার মধ্যে অবজেক্ট ওরিয়েন্টেড এপ্লিক্যশান লেয়ার এর ধারণাটি ছিল অসাধারণ যাতে কিনা অবজেক্ট ধরে ধরে কাস্টমাইজড সফটওয়্যার তৈরি করে ফেলা যায়। Patrick Naughton ইতিমধ্যে NeXT এর দিকে যাওয়ার জন্য মনস্থির করে ফেলেছেন কিন্তু তখন একবার তাকে শেষ সুযোগ হিসেবে একটি অতি গোপন প্রজেক্টের অনুমোদন দেওয়া হয় যার কথা কেউ জানতো না। কিছুদিন পরেই তার সাথে যুক্ত হয় James Gosling এবং Mike Sheridan । তখন এর নাম দেওয়া হয় গ্রিন প্রজেক্ট। সময়ের সাথে গ্রিন প্রজেক্ট এর দস্তোদ্গম হয় এবং তারা কম্পিউটার ছাড়াও বিভিন্ন ডিভাইস নিয়ে নার্চার করতে থাকে।

এর মধ্যে ১৩ জন স্টাফ এই গ্রিন টিম ক্যালিফোর্নিয়ার মেনলো পার্কের সেন্ড হিল রোড এর একটি ছোট্ট অফিসে কাজ করতে থাকে। তাদের প্রধান উদ্দেশ্য সি প্লাস প্লাস এর একটি ভাল ভার্সন তৈরি করা যা কিনা হবে অনেক দ্রুতগামী এবং রেস্পন্সিভ। সেই সময়ে কম্পিউটার ছাড়াও কনজুমার ইলেকট্রনিক্স যেমন - পিডিএ, Cable-Set Top Box ইত্যাদির চাহিদা বেড়ে গেছে। একদল ইঞ্জিনিয়ার এক সাথে থাকলে যা হয়, তারা নানারকম জিনিস নিয়ে চিন্তা করতে থাকে,

নানা রকম আইডিয়া তৈরি হয়, তা থেকে প্রোটোটাইপ তৈরি করতে থাকে। এর মধ্যে জেমস গসলিং তার সি প্লাস প্লাস এনহান্সমেন্ট চালিয়ে যেতে থাকেন। তিনি এর নাম দেন সি প্লাস প্লাস প্লাস প্লাস মাইনাস মাইনাস (C++ ++ - -) । এখানে বাড়তি ++ মানে হচ্ছে নতুন জিনিস যোগ করা এবং - - মানে হচ্ছে কিছু জিনিস ফেলে দেওয়া। জেমস গসলিং এর জানালা দিয়ে একটি ওক গাছ দেখা যায়। একদিন তিনি অফিস থেকে বের হয়ে ঐ গাছটির নিচে দাড়ান এবং সাথে সাথে C++ ++ - - নাম পরিবর্তন করার সিদ্ধান্ত নেন এবং নতুন নাম দেন ওক।

এর মধ্যে ইঞ্জিয়াররা মিলে এম্বেডেড সিস্টেম নিয়ে নার্চার করতে থাকা অবস্থায় নানা রকম সমস্যার সম্মুখীন হন। এম্বেডেড সিস্টেম এ মেমরি কম থাকে, প্রসেসিং পাওয়ার ও কম থাকে। এই সিস্টেমে সি++ (যা কিনা কম্পিউটার এর মতো বড় ফ্রুটিপ্রিন্টের হার্ডওয়্যারের জন্যে ডিজাইন করা) চালাতে গিয়ে তারা অদ্ভুত অদ্ভুত সমস্যার সম্মুখীন হতে থাকে। এইসব সমস্যার সমাধান করার জন্যে গ্রিন টিম নানা রকম চিন্তা ভাবনা করতে থাকে। এই সময়ে মানুষ পিডিএ, Cable-Set Top Box গুলোর মরণদশা দেখতে শুরু করে। কারণ যদিও ওক নিয়ে যথেষ্ট এগিয়েছে কিন্তু এটি কোনভাবেই এদেরকে সাহায্য করতে পারছিল না। একমাত্র একটি অলৌকিক ঘটনায় পারে এই প্রজেক্ট সফল করতে। ঠিক তখনি সেই প্রতীক্ষিত প্রত্যাশা আলোর মুখ দেখে। জেমস জেমস গসলিং আউট অব দ্যা বক্স একটা যুগান্তকারী ধারণা নিয়ে আসে। সেটি হলো ভার্চুয়াল মেশিন। অর্থাৎ আমরা একটা কাল্পনিক মেশিনের জন্যে কোড লিখবো যা কিনা কম্পাইল হয়ে একটি অন্তর্বর্তীকালীন কোড তৈরি করবে। এবং জাভা ভার্চুয়াল মেশিন সেই অন্তর্বর্তীকালীন কোডকে রান টাইম-এ রিয়েল ডিভাইসের জন্যে প্রয়োজন অনুযায়ী মেশিন কোড তৈরি করবে।

ঠিক সেই সময়েই National Center for Supercomputing Applications (NCSA) একটি কমার্শিয়াল ওয়েব ব্রাউজার বের করে এবং তাদের টিম ইন্টারনেট এর ভবিষ্যৎ নিয়ে ভাবতে শুরু করে। তারা একটি নতুন ধারণা নিয়ে আসে সেটি হলো, একধরণের ছোট্ট প্রোগ্রাম যা কিনা ব্রাউজার এর মধ্যে চলবে - এর নাম দেয় অ্যাপলেট। অ্যাপলেট ধারণা থেকে তারা ঠিক করে অ্যাপলেট এর জন্যে কিছু স্ট্যাডার্ড – এটি হতে হবে ছোট্ট, খুব সিম্পল, এর স্ট্যাডার্ড এপিআই থাকতে হবে, এটি হবে প্লাটফর্ম ইন্ডিপেন্ডেন্ট, এবং আউট-অব-দ্যা বক্স নেটওয়ার্কিং প্রোগ্রামিং করা যাবে। তারা তখনকার সময়ের ইন্টারনেট বুমকে উদ্দ্যেশ্য করে নেক্সট জেনারেশান প্রোডাক্ট ডেভেলপ করতে চেয়েছিল। এই প্রজেক্ট এর কার্টুন নাম ছিল Duke (যা কিনা এখন জাভা-এর মাস্কট হিসেবে চিনি)। কিন্তু সমস্যা হচ্ছে এর কোনটিই ঠিক মতো সি++ দিয়ে করা যাচ্ছিল না। সুতরাং পরবর্তীতে তারা সিদ্ধান্ত নেয় যে এমবেডেড সিস্টেমের সমস্যার সমাধানটি তারা ওয়েব ব্রাউজার এর ক্ষেত্রেও ব্যবহার করবে । সেই সময়ে মানুষ ওয়েব ব্রাউজার এর শুধুমাত্র স্ট্যাটিক পেইজ এ টেক্সট আর ইমেজ ছাড়া কিছু দেখতে পেত না। এই টেকনোলজি ব্যবহার করায় ব্রাউজার এনিমেশান থেকে শুরু করে ইন্টারেকটিভ অ্যাপলেট সকলের নজর কাড়ে যা কিনা জাভা প্রোগ্রামিং ল্যাংগুয়েজ এর সফলতার মূল কারণ।

জেমস গসলিং এর এই ভার্চুয়াল মেশিন-এর সল্যুশান ছিল সত্যিকার অর্থেই যুগান্তকারী এবং `গ্রিন টিম` এর রিলিজ দিতে প্রস্তুত। কিন্তু তখন-ই নতুন ঝামেলার সূচনা হয়, lawyers এসে তাদের জানায় এর নাম Oak দেওয়া যাবে না, কারণ এটি ইতিমধ্যেই Oak Technologies এর ট্রেড মার্ক। সুতরাং নাম পরিবর্তন করতে হবে। শুরু হয় ব্রেইনস্টর্মিং । কিন্তু কোন ভাবেই একটি ভাল নাম নির্বাচন করা যাচ্ছিল না। অনেকেই অনেক ধরণের নাম উপস্থাপন করে, যেমন - DNA, Silk , Ruby, yuck, Silk, Lyric, Pepper, NetProse, Neon, Java ইত্যাদি ইত্যাদি। এর সব গুলো লিগাল ডিপার্টমেন্ট এ সাবমিট করার পর মাত্র Java, DNA, and Silk এই তিনটি নাম ফিরে আসে যা কিনা ক্লিন। নাম নিয়ে ঘণ্টার পর ঘণ্টার মিটিং চলতে থাকে। এর মধ্যে Chris Warth প্রপোজ করে Java, কারণ তখন তার হাতে ছিল এক কাপ গরম Peet's Java (কফি)। শেষ পর্যন্ত নাম ঠিক করা হয় Java কারণ একমাত্র এই নামেই সব থেকে পজিটিভ রিএকশান পাওয়া যাচ্ছিল।

১৯৯৫ সালের মে মাসে জাভা এর প্রথম পাবলিক ভার্সন রিলিজ হয়।

এর পরের ইতিহাস আমরা সবাই জানি। জাভা হচ্ছে এই গ্রহের সবচেয়ে সফল প্রোগ্রামিং ভাষা।

তোমার প্রথম জাভা প্রোগ্রাম

আমরা এই চ্যাপ্টার এ যে যে বিষয়গুলো দেখবো সেগুলো হলো-

- প্রোগ্রামিং ল্যাংগুয়েজ কি এবং কেন
- কেন জাভা
- জাভা কিভাবে কাজ করে, ভেতরের বৃত্তান্ত
- জাভা একটি কম্পাইল্ড ল্যাংগুয়েজ না ইন্টারপ্রেটেড ল্যাংগুয়েজ
- জাভা ভার্চুয়াল মেশিন কি এবং কিভাবে কাজ করে
- জাভা রানটাইম
- জাভা ডেভেলপমেন্ট কিট এবং আইডিই
- জেডিকে ইনস্টলেশন
- একটি হ্যালো ওয়ার্ল্ড প্রোগ্রাম

প্রোগ্রামিং ল্যাংগুয়েজ কি ?

প্রোগ্রামিং ল্যাংগুয়েজ হচ্ছে এক ধরণের কৃত্রিম ভাষা যা কিনা যন্ত্র বিশেষ করে কম্পিউটার-এর আচরণ নিয়ন্ত্রণ করার জন্যে ব্যবহার করা হয়। মানুষের ভাষার মতো এর কিছু সিনট্যাক্স এবং সেম্যান্টিক্স অর্থাৎ নিয়মকানুন ও অর্থ থাকে। আমাদের এই বই এর উদ্দেশ্য হচ্ছে একটি বিশেষ ভাষার(জাভা) নিয়মকানুন গুলো জেনে নেওয়া। সুতরাং পড়তে থাকুন।

কেন জাভা?

পৃথিবীতে এখন পর্যন্ত অনেক গুলো প্রোগ্রামিং ভাষা তৈরি করা হয়েছে। এদের প্রত্যেকটির উদ্দেশ্য ভিন্ন ভিন্ন। http://en.wikipedia.org/wiki/List_of_programming_languages এখানে একটি প্রোগ্রামিং ল্যাংগুয়েজ এর একটি লিস্ট দেওয়া আছে- দেখে নেওয়া যেতে পারে। প্রত্যেকটি ল্যাংগুয়েজ এর কিছু সুবিধা অসুবিধা আছে, এবং ল্যাংগুয়েজ গুলো প্রতিনিয়ত উন্নত হচ্ছে, এবং নতুন নতুন ল্যাংগুয়েজ তৈরি হচ্ছে।

যে যে কারণে জাভা শেখা যেতে পারে এখন সেগুলো নিয়ে আলোচনা করা যাক-

- এটি খুব-ই (Readable)পাঠযোগ্য, সহজে বুঝা যায়। অন্য যে কোন প্রোগ্রামিং ব্যাকগ্রাউন্ড এর প্রোগ্রামার খুব সহজেই একটি জাভা-ফাইল দেখে বুঝতে পারবে আসলে কোড এ কি লেখা আছে।
- সি কিংবা সি++ এ কোড করার সময় আমাদের অনেক সময়-ই লিংকিং, অপটিমাইজেশান, মেমরি এলোকেশান, মেমরি ডি-এলোকেশান, পয়েন্টার ডিরেফারেন্সিং ইত্যাদি নানা রকম জিনিস নিয়ে ভাবতে হয়, কিন্তু জাভার ক্ষেত্রে এগুলোর কথা ভাবতেই হয় না। খুব বেশি চিন্তা না করে আমরা নিশ্চিতভাবে জাভা কম্পাইলার এর উপর সব কিছু ছেড়ে দিতে পারে।
- জাভাতে অসংখ্য API আছে যেগুলো খুবই স্টেবল, খুব বেশি চিন্তাভাবনা না করেই এদের নিয়ে খুব সহজেই কাজ করে ফেলা যায়।
- জাভা -র সব কিছুই ওপেন সোর্স।
- জাভা ভার্চুয়াল মেশিন সম্ভবত সফটওয়্যার- জগতে সব থেকে চমৎকার সৃষ্টি। জাভা-এর সাথে এর আরও অনেকগুলো ল্যাংগুয়েজ যেমন- গ্রুভি, স্ক্যালা ইত্যাদি নিয়ে কাজ করা যায়।
- গত ১৫ বছরে চমৎকার অনেকগুলো ডেভেলপমেন্ট এনভায়রনমেন্ট তৈরি হয়েছে যেগুলো খুবই ইন্টেলিজেন্ট – যেমন- Eclipse, IntelliJ IDEA, netbeans etc. । এগুলো মাধ্যমে খুব আয়েশের সাথেই কোড করা যায়, ডিবাগ করা যায়।
- এটি একটি অবজেক্ট ওরিয়েন্টেড- টাইপ সেইফ প্রোগ্রামিং ল্যাংগুয়েজ।
- এটি পোর্টেবল যে কোন প্লাটফর্মে চলে। একবার কোড লিখে সেটি যে কোন মেশিনে(উইন্ডোজ , লিনাক্স , ম্যাক) চালানো যায়।
- অনেক বড় কমিউনিটি সাপোর্ট- সারা দুনিয়াতে মিলিয়নস অব জাভা প্রোগ্রামার ছড়িয়ে ছিটিয়ে আছে।
- এটির পারফরমেন্স নিয়ে বলা চলে কোন সন্দেহ নেই।
- ইন্ডাস্ট্রি গ্রেডেড, বড় বড় এন্টারপ্রাইজ অ্যাপ গুলো সাধারণত জাভা দিয়ে লেখা হয়।
- এটি পৃথিবীতে দ্বিতীয় জনপ্রিয় ল্যাংগুয়েজ- http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html

এই লিস্ট এখানেই থামিয়ে দেই- কারণ এটি শেষ হতে চাইবে না কখনোই।

জাভা কিভাবে কাজ করে ?

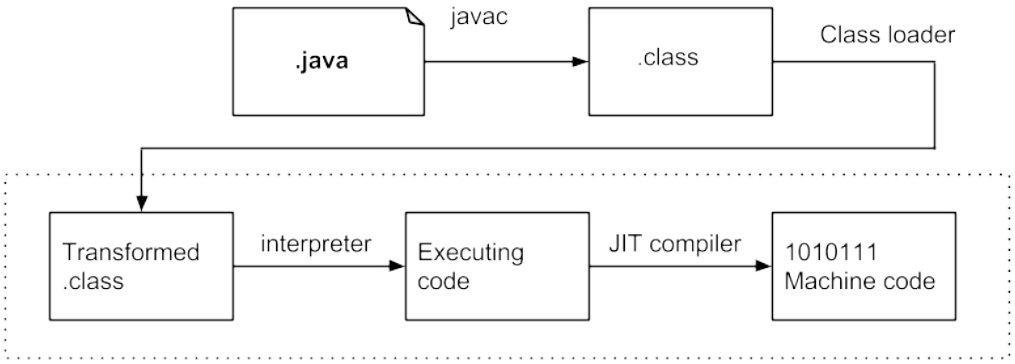


Figure: How java works

জাভা কোডকে কম্পাইল করলে সেটি একটি অন্তর্বর্তীকালীন ল্যাংগুয়েজ এ রূপান্তরিত হয়। এটি ঠিক হিউম্যান রিডএবল না আবার মেশিন রিডএবল ও না। একে আমরা বলি বাইট কোড। এই বাইটকোড শুধুমাত্র জাভা ভার্চুয়াল মেশিন(JVM) বুঝতে পারে। JVM বাইট কোড কে ইন্টারপ্রেট করে মেশিন ল্যাংগুয়েজ এ রূপান্তরিত করে। এর জন্যে JVM জাস্ট ইন টাইম(JIT) কম্পাইলার ব্যবহার করে। সুতরাং দেখা যাচ্ছে, জাভা কোডকে প্রথমে কম্পাইল করা হয়, তারপর সেই আউটপুট কে ইন্টারপ্রেট করা হয়। এক্ষেত্রে প্রশ্ন হতে পারে, জাভা আসলে কি? কম্পাইল্ড ল্যাংগুয়েজ নাকি ইন্টারপ্রেটেড ল্যাংগুয়েজ? উত্তর হচ্ছে জাভা একি সাথে দুটোই।

উপরের বর্ণনা থেকে আমরা তিনটি জিনিস জানলাম -

১. বাইট কোড – এটি হচ্ছে এক ধরণের ইন্ট্রাকশান সেট- যা কিনা শুধুমাত্র জাভা ভার্চুয়াল মেশিন বুঝতে পারে। জাভা কোড (হিউম্যান রিডএবল) অর্থাৎ আমরা যে কোড গুলো লিখবো সেগুলো কে জাভা কম্পাইলার দ্বারা কম্পাইল করলে বাইটকোড তৈরি হয়। এই বাইটকোড গুলো .class এক্সটেনশন যুক্ত বাইনারী ফাইলে স্টোর করা হয়।

২. জাভা ভার্চুয়াল মেশিন**(JVM)** - এটি মূলত একটা বাস্তব মেশিনের ভেতর একটা কাল্পনিক মেশিন। সহজ কথায়- এটি একটি সফটওয়্যার যা কিনা বাইট কোড পড়ে সেগুলো মেশিন এক্সিকিউটেবল কোড-এ রূপান্তরিত করতে পারে। JVM অনেকগুলা মেশিনের জন্যে লেখা হয়েছে- অর্থাৎ এটি উইন্ডোজ, ম্যাক OS, লিনাক্স, আইবিএম mainframes, সোলারিস ইত্যাদি অপারেটিং সিস্টেমের জন্যে আলাদা আলাদা করে লেখা হয়েছে। এর ফলে, আমরা যদি একবার কোন জাভা প্রোগ্রাম লিখি, সেটি যেকোন মেশিনে চালানো যাবে। এর কারণ আমরা এখন কোন নির্দিষ্ট মেশিনকে উদ্দেশ্য না করে শুধু মাত্র JVM কে উদ্দেশ্য করে কোড লিখি । যেহেতু সব মেশিনের জন্যেই JVM আছে, সুতরাং আমাদের কোড সব মেশিনেই চলবে। আর এভাবেই - **“Write once, run anywhere”** বা **WORA** সম্ভব হয়েছে।

৩. জাস্ট ইন টাইম**(JIT)** কম্পাইলার – এটি মূলত JVM এর একটি অংশ। আমরা যে জাভা কোড কম্পাইল করার সময় তৈরি করি সেগুলো মূলত JIT কম্পাইলার প্রসেস করে। একে dynamic translator ও বলা যায়- কারণ এটি রানটাইম-এ অর্থাৎ প্রোগ্রাম চলাকালিন সময়ে বাইটকোড প্রসেস করে।

এবার আমরা আরও কিছু টার্মিনোলোজি(পরিভাষা) এর সাথে পরিচিত হই।

জাভা রানটাইম এনভায়রনমেন্ট **(JRE)** –এটি মূলত একটি জাভা প্রোগ্রাম রান করার জন্যে অন্তত:পক্ষে যে সব কম্পোনেন্ট লাগে তার একটি প্যাকেজ। এর মধ্যে থাকে JVM এবং কিছু স্ট্যান্ডার্ড এপিআই।

জাভা ডেভেলপার কিট **(JDK)** – এটি হচ্ছে JRE এবং জাভা কোড লেখার জন্যে যে সব টুল গুলো লাগে তার একটি সেট। জাভা প্রোগ্রাম লেখার জন্য শুধু মাত্র JDK থাকলেই চলে কারণ এর মাঝেই সব কিছু দেয়া থাকে।

জাভার তিনটি সাবসেট আছে সেগুলো হলো -

জাভা স্ট্যান্ডার্ড এডিশন **(JSE)**

- ডেস্কটপ এবংস্টস্ট্যান্ড-অ্যালোন সার্ভার এপ্লিকেশান তৈরি করার জন্যে যে সব টুল এবং এপিআই দরকার হয় সেগুলোকে আলাদা করে এর নাম দেওয়া হয়েছে জাভা স্ট্যান্ডার্ড এডিশন।

জাভা এন্টারপ্রাইস এডিশন **(JEE)** – এটি JSE এর উপর তৈরি ওয়েব এবং অনেক বড় মাপের এন্টারপ্রাইজ এপ্লিকেশান তৈরি করার জন্যে যে সব কম্পোনেন্ট দরকার হয় সেগুলোকে আলাদা করে এর নাম দেওয়া হয়েছে জাভা এন্টারপ্রাইস এডিশন- উদারহরণসরূপ এর কম্পোনেন্ট গুলো হচ্ছে-

- Servlets

- Java Server Pages (JSP)
- Java Server Faces (JSF)
- Enterprise Java Beans (EJB)
- Two-phase commit transactions
- Java Message Service message queue API's (JMS)
- etc.

জাভা মাইক্রো এডিশন **(JME)**

- এটি মূলত জাভা স্ট্যান্ডার্ড এডিশন এর সংক্ষিপ্ত এডিশন। ইন্টারনেট অব থিংস, এমবেড ডিভাইস, মোবাইল ডিভাইস, মাইক্রোকন্ট্রোলার, সেন্সর, গেটওয়ে, মোবাইল ফোন, ব্যক্তিগত ডিজিটাল সহায়ক (পিডিএ), টিভি সেট টপ বক্স, প্রিন্টার ইত্যাদি জন্যে তৈরি জাভার এই সংক্ষিপ্ত এডিশন কে বলা হয় - জাভা মাইক্রো এডিশন ।

এবার তাহলে জাভা চলুন জাভা ইন্সটল করে ফেলি--

লিনাক্স মেশিনে জাভা ইনস্টল করতে নিচের ধাপ গুলো apply করতে হবে-

- ধাপ ১: নিচের লিংক থেকে জাভা ডাউনলোড করে নিন।

Oracle JDK 7 Download Link

- ধাপ ২: এরপর টার্মিনাল থেকে যেখানে জাভা ডাউনলোড হয়েছে সেখানে যান-

```
cd ~/Download
```

- ধাপ ৩: এবার JDK ইনস্টল করি-

```
sudo tar -xzvf jdk-7u21-linux-i586.tar.gz --directory=/usr/local/
```

```
sudo ln -s /usr/local/[jdk_folder_name]/ /usr/local/jdk
```

jdk_folder_name - আপনার পছন্দমত একটি নাম দিন।

- ধাপ ৪: আবার টার্মিনালে ফিরে যান- .bashrc অপেন করুন।

```
sudo gedit .bashrc
```

- ধাপ ৫ : .bashrc ফাইল-এ নিচের লাইনটি এড করুন।

```
export JAVA_HOME=/usr/local/jdk
```

Save and close .bashrc file.

- ধাপ ৬: কম্পাইল .bashrc ফাইল

```
source .bashrc
```

- ধাপ ৭: এবার পরীক্ষা করে দেখা যাক জাভা ইনস্টল হয়েছে কিনা। আবার টার্মিনাল ওপেন করুন এবং নিচের লাইনটি টাইপ করুন।

```
java -version
```

যদি সবকিছু ঠিকঠাক থাকে তাহলে আপনি নিচের তথ্য গুলো দেখতে পারবেন-

```
java version "1.7.0_65"
Java(TM) SE Runtime Environment (build 1.7.0_65-b17)
Java HotSpot(TM) 64-Bit Server VM (build 24.65-b04, mixed mode)
```

আর উইন্ডোজ মেশিনের ক্ষেত্রে এটি আরো সহজ। এর জন্যে শুধুমাত্র JDK টি ডাউনলোড করে ডাবল-ক্লিক করেই এটি ইনস্টল করা যাবে।

IDE-

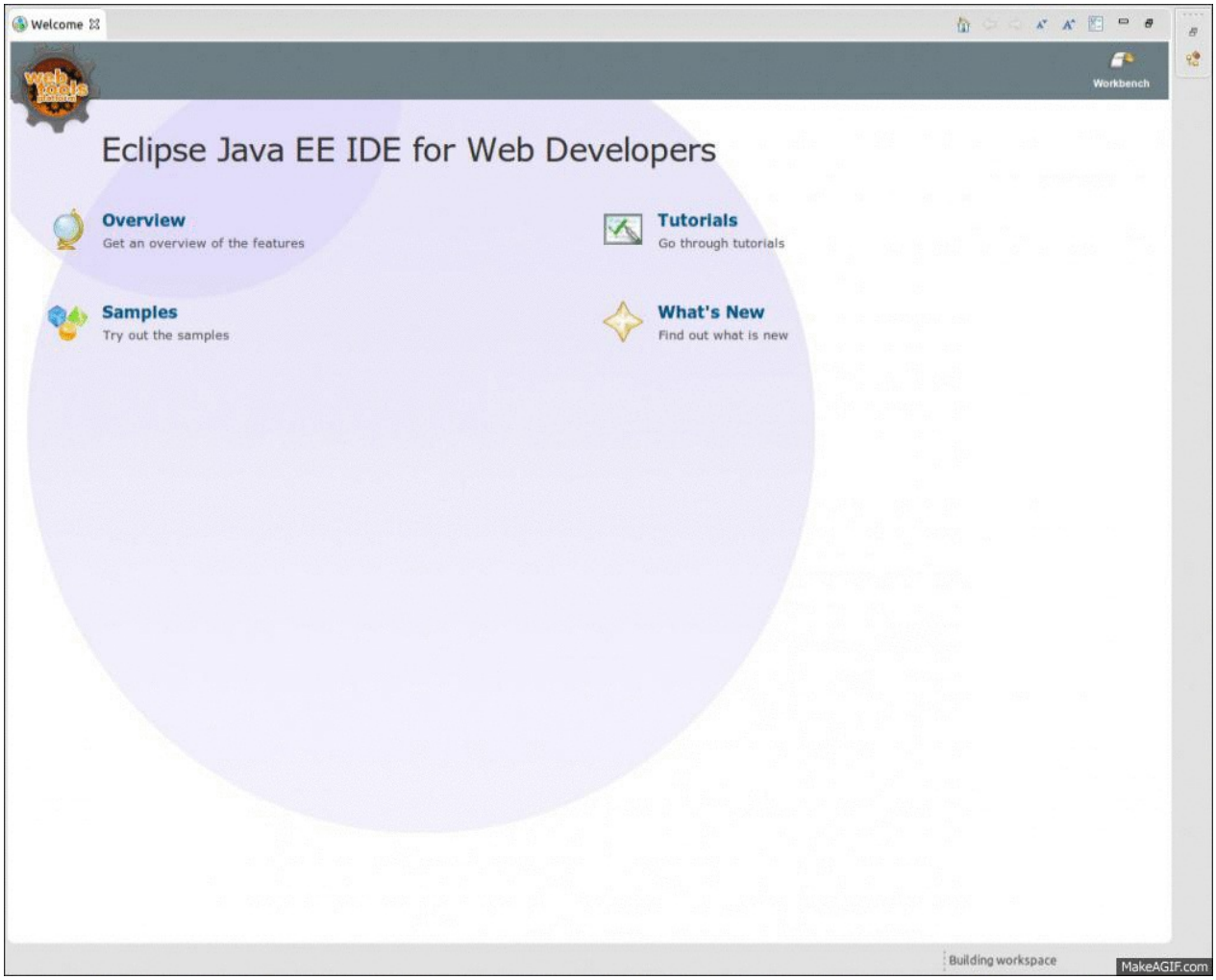
এক্ষেত্রে আমি দুটি আইডিইর কথা বলতে পারি-

- ১. Eclipse - https://www.eclipse.org/downloads/
- ২. IntelliJ IDEA - http://www.jetbrains.com/idea/download/

তবে এই টিউটোরিয়ালে আমরা Eclipse ব্যবহার করবো।

তো চলুন- এবার তাহলে আমাদের প্রথম Hello world প্রোগ্রামটি লিখে ফেলি।

```
package bd.com.howtocode.java.helloworld;

public class HelloWorld {

    public static void main(String[] args) {
        System.out.println("Hello, world!");
    }
}
```



পাঠ ২: সিনট্যাক্স

- প্যাকেজ ডিক্লেয়ারেশান
- ইম্পোর্ট
- ক্লাস
- ফিল্ডস
- মেথডস
- কন্সট্রাকটরস
- কমেন্টস

এই চ্যাপ্টারে আমি একটি জাভা প্রোগ্রাম এর মৌলিক কিছু ওভারভিউ দেয়ার চেষ্টা করবো। তবে শুরুতে সুবিধার্থে আমাদের কিছু টার্মস সম্পর্কে জেনে নেওয়া জরুরি।

অবজেক্ট

যেহেতু জাভা একটি অবজেক্ট ওরিয়েন্টেড ল্যাংগুয়েজ, সুতরাং শুরুতে জানতে হবে অবজেক্ট কি। অবজেক্ট এর মানে আমারা যা জানি, সেটা হচ্ছে আমাদের জড়জগতের কোন বস্তু, যাকে ঠিক স্পর্শ করা যায়। তবে যেহেতু আমরা কল্পনা করতে পারি, আমরা অনেক কিছু ধরে নিতে পারি, মনে করুন - একটি বাইসাইকেল। বাইসাইকেল বলতেই আমদের মাথায় একটি চিত্র চলে আসে। আমরা এর বৈশিষ্টগুলো জানি, যেমন এটির দুইটি চাকা থাকে, একটি বসার সিট থাকে, এর ব্রেক আছে। তারপর এও জানি যে এটি কি করে, অর্থাৎ সাইকেল এর কাজ গুলোও আমরা জানি- যেমন এটি চলে। দেখা যাচ্ছে যে আমরা একটি বাইসাইকেল এর অবস্থা ও আচরণ সম্পর্কে জানি। এই অবস্থা ও আচরণ গুলো নিয়েই বাইসাইকেল একটি অবজেক্ট।

আমরা যদি আমাদের কল্পনাটুকু আরেকটু বাড়িয়ে নিয়ে বলি, সাইকেল হচ্ছে একটি সফটওয়্যার কম্পোনেন্ট যা কিনা কম্পিউটারে চলে, আমার মনে হয় কারো আপত্তি থাকার কথা নয়।

যেহেতু আমারা প্রোগ্রামিং নিয়ে আলোচনা করছি, সুতরাং এভাবে বলি, আমরা যদি একটা প্রোগ্রাম লিখি, সেই প্রোগ্রামের ছোট্ট একটি অংশ যার আমাদের এই বাইসাইকেল এর মতো বৈশিষ্ট্য থাকে, এবং একটি কিছু কাজ সম্পাদন করতে পারে, তাহলে সেই ছোট্ট অংশটিকে অবজেক্ট বলতে পারি।

ক্লাস

মনে করি আমরা একটা বাড়ি বানাতে চাই। প্রথমে আমরা চিন্তা করি বাড়িটা আসলে কিভাবে বানাবো। আমরা জায়গা নির্বাচন করি। তারপর চিন্তা করি বাড়িটি কত-তলা হবে, কয়টা এপার্টমেন্ট হবে, এপার্টমেন্ট গুলো কত স্কয়ারফিটের হবে। তারপর চিন্তা করি, একটা এপার্টমেন্ট এ কয়টি রুম হবে, ড্রয়িং রুমের দৈর্ঘ্য কত হবে, কয়টা বাথ থাকবে, বেলকনি কোথায় থাকবে, রান্না ঘর কোথায় হবে ইত্যাদি ইত্যাদি। আচ্ছা এগুলো ঠিক হয়ে গেল, এখন আমরা চিন্তা করবো আরও জটিল কাজ নিয়ে। ওয়্যারিং নিয়ে, প্রত্যেক রুমে কয়টা পয়েন্ট থাকবে, পানির লাইন কিভাবে নেব। তারপরে বাথরুমে কি ধরণের টাইল ব্যবহার করবো, ফ্লোরে কোন গুলো।

অর্থাৎ বাড়িটি বানানোর আগেই আমরা সব কিছু নির্ধারণ করে ফেলছি এবং আমরা এই বিষয়গুলো সব লিপিবদ্ধ করে রাখি। তারপর এই লিপিবদ্ধ লেখাগুলোকে নানাভাবে পরীক্ষা করে ক্রস চেক করে চূড়ান্ত করি। এর একটি গলাভরা নাম আছে, সেটা হচ্ছে- blueprint.

আমাদের এক্ষেত্র বাড়টি হচ্ছে অবজেক্ট। এই অবজেক্ট বানানোর আগে আমাদের blueprint এর দরকার হয়। আর এই blueprint কেই আমরা বলি ক্লাস।

আমরা তাহলে এখন অবজেক্ট এবং ক্লাস এর ধারণা জানি। এবার তাহলে আমাদের মূল বিষয় সিনট্যাক্স নিয়ে কথা বলি-

আমরা যারা সি কিংবা অন্য কোন প্রোগ্রামিং ল্যাংগুয়েজ আগে থেকেই জানি, একটি প্রোগ্রামে দুটি জিনিস অবশ্যই কমন থাকে - সেগুলো হলো - ফাংশান এবং ডেটা ।

একটি জাভা প্রোগ্রাম লিখতে হলে আমাদেরকে অবশ্যএই একটি ফাইল তৈরি করতে হবে যার এক্সটেনশন হবে .java. উদাহরণসরূপ- HelloWorld.java
এবার আমরা লক্ষ্য করি একটি জাভা প্রোগ্রামে কি কি থাকে-

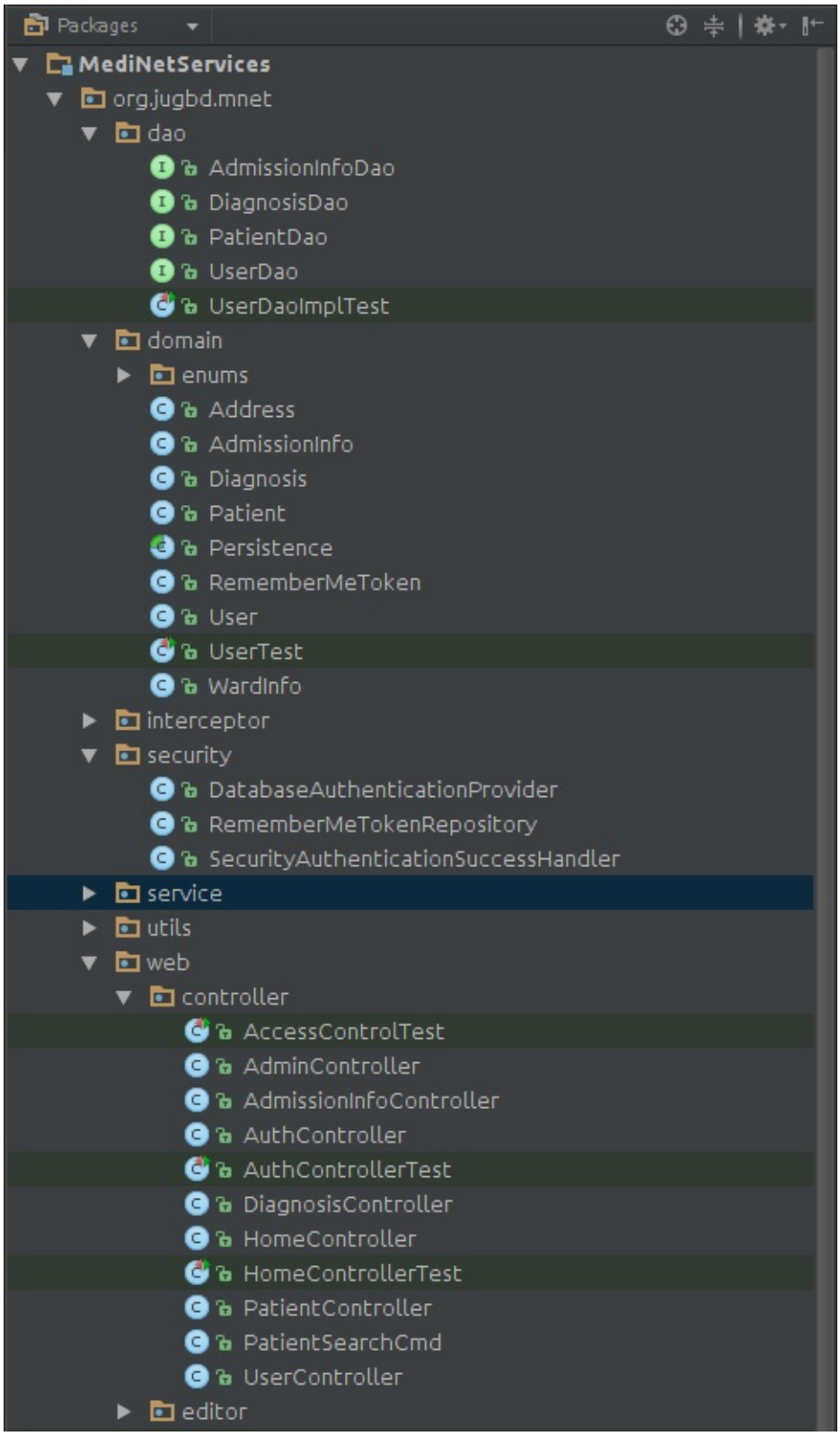
- প্যাকেজ ডিক্লারেশন
- ইম্পোর্ট স্টেটমেন্টস
- টাইপ ডিক্লারেশন
    - ফিল্ডস
    - মেথডস

উপরের নামগুলো নিয়ে দ্বন্দ্ব লাগলে সমস্যা নেই, এক্ষণি সেগুলো নিয়ে আলোচনা করছি, তবে তার আগে একটি জাভা প্রোগ্রাম দেখে নিই।

```java
package bd.com.howtocode.java.tutotorial.syntax;

import java.util.HashMap;

public class HelloWorld {
    protected final String hello = "value";

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

এই কোডটির শুরুতেই আছে প্যাকেজ ডিক্লারেশন। আমরা আমাদের কম্পিউটারে নানা ধরণের ফাইল বিভিন্ন ফোল্ডারে সাজিয়ে রাখি। যেমন- মুভি ফোল্ডারে হয়তো আমরা শুধুই মুভি রাখি, সেখানে অন্য ফাইল রাখি না। আবার মুভি ফোল্ডারে এর মধ্যে আরো সাব-ফোল্ডার তৈরি করি আরো আলাদা করার জন্যে, যেমন – বাংলা মুভি, ইংরেজি মুভি ইত্যাদি। জাভাতে প্যাকেজ বলতে এই ধারণটিই বুঝায়। একটি জাভা প্রোগ্রামিং ভাষায় লেখা সফটওয়্যার এ শত শত বা হাজার হাজার পৃথক ক্লাস থাকতে পারে। এজন্যে প্যাকেজ ডিক্লারেশন এর মাধ্যমে আমরা একি রকম ক্লাস গুলো একটি প্যাকেজের মধ্যে আলাদা করে রাখি।

উদাহরণসরূপ এখানে প্যাকেজ স্ট্রাকচার এর একটি স্ক্রিনশট দেওয়া হল-



প্যাকেজ নাম গুলাকে লোয়ার কেস অক্ষরে-এ লিখতে হয়।

কোম্পানি গুলো তাদের ইন্টারনেট ডোমেইন নেইম কে উল্টো করে তাদের প্যাকেজের নাম লিখে। যেমন - example.com এর একটি প্রোগ্রামার একটি প্যাকেজের নাম লিখবে এইভাবে- com.example.package.

আমাদের ক্ষেত্রে-

```
package bd.com.howtocode.java.tutotorial.syntax;
```

তারপর আমাদের প্রোগ্রামের দ্বিতীয় লাইনটি হলো - ইম্পোর্ট স্টেটমেন্টস। অন্য কোন প্যাকেজের ক্লাস যদি আমাদের প্রোগ্রামে দরকার হয় তাহলে আমারা সেটিকে এভাবে ইম্পোর্ট করতে পারি। এটি সি প্রোগ্রামিং এর ইনক্লুড স্টেটমেন্টস এর মতো।

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

এর পরের লাইনটি হলো টাইপ ডিক্লারেশন। জাভাতে একটি টাইপ একটা ক্লাস অথবা ইন্টারফেইস অথবা এনাম হতে পারে(ইন্টারফেইস এবং এনাম নিয়ে পরে আলোচনা করা হবে)। ক্লাস ক্ষেত্রে শুরুতে class কিওয়ার্ড লিখেতে হয় তারপর কার্লি ব্রেস { শুরু এবং শেষ } করতে হয়। আমাদের পরবর্তি প্রতিটা লাইন কোড এই কার্লি ব্রেস { } এর ভেতরে লিখতে হবে।

```
public class HelloWorld { }
```

এখানে অতিরিক্ত একটি public কিওয়ার্ড দেখা যাচ্ছে। এই মুহূর্তে শুধু মনে রাখুন ক্লাস এর শুরুতে এটি লিখতে হয়। পরে এটি নিয়ে আলোচনা করা হবে।

এর পরেই আমরা যা দেখছি তাকে বলা হয় ফিল্ড ডিক্লারেশন। অর্থাৎ আমরা যে বিভিন্ন রকম ভ্যারিয়বল ডিক্ল্যার করি, সেগুলো।

```
    protected final String hello = "value";
```

এবং এর পরেই থাকে মেথড। সি কিংবা অন্যান্য প্রোগ্রামিং ল্যাংগুয়েজ এ যাকে আমরা ফাংশন কিংবা সাবরুটিন বলে থাকে, এখানে আমরা সেগুলোকে মেথড বলি।

এক্ষেত্রে আমাদের মেথড হচ্ছে -

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

এটি হচ্ছে মেইন মেথড। জাভা প্রোগ্রামকে রান করতে হলে অবশ্যই কোন ক্লাসে একটি মেইন মেথড থাকতে হবে। এবার আমরা কিছু জিনিস প্রিন্ট করার চেষ্টা করি-

জাভাতে কনসলে কিছু প্রিন্ট করার জন্যে System.out.println() অথবা System.out.print() ব্যবহার করা হয়।

আমরা যদি নিচের প্রোগ্রমটি রান করি-

```
package bd.com.howtocode.java.tutotorial.syntax;

public class HelloWorld {
    public static void main(String[] args) {
        System.out.println("Hello, world!"); // Advance the cursor to the beginning of next line after printing
        System.out.println();               // Print a empty line
        System.out.print("Hello, world!");   // Cursor stayed after the printed string
        System.out.println("Hello,");
        System.out.print(" ");              // Print a space
        System.out.print("world!");
        System.out.println("Hello, world!");
    }
}
```

তাহলে কনসলে নিচের লাইন গুলো প্রিন্ট হবে-

```
Hello, world!

Hello, world!Hello,
 world!Hello, world!
```

আমরা ইতিমধ্যে জানি ক্লাস কি- তাহলে এবার একটি ক্লাস লিখে ফেলা যাক-

```
    package bd.com.howtocode.java.tutotorial.syntax;

    /**
```

```
 * @author Bazlur Rahman Rokon
 * @since 9/20/14.
 */
public class Bicycle {
    int cadence = 0;
    int speed = 0;
    int gear = 1;

    void changeCadence(int newValue) {
        cadence = newValue;
    }

    void changeGear(int newValue) {
        gear = newValue;
    }

    void speedUp(int increment) {
        speed = speed + increment;
    }

    void applyBrakes(int decrement) {
        speed = speed - decrement;
    }

    void printStates() {
        System.out.println("cadence:" +
                cadence + " speed:" +
                speed + " gear:" + gear);
    }
}
```

আমরা ক্লাস এবং অবজেক্ট কি জানি, কিন্তু কিভাবে ক্লাস থেকে অবজেক্ট তৈরি করতে হয় সেটি এবার দেখা যাক-

```
package bd.com.howtocode.java.tutotorial.syntax;

/**
 * @author Bazlur Rahman Rokon
 * @since 9/20/14.
 */

public class BicycleDemo {
    public static void main(String[] args) {
        // Create two different
        // Bicycle objects

        Bicycle bike1 = new Bicycle();
        Bicycle bike2 = new Bicycle();

        // Invoke methods on
        // those objects
        bike1.changeCadence(50);
        bike1.speedUp(10);
        bike1.changeGear(2);
        bike1.printStates();

        bike2.changeCadence(50);
        bike2.speedUp(10);
        bike2.changeGear(2);
        bike2.changeCadence(40);
        bike2.speedUp(10);
        bike2.changeGear(3);
        bike2.printStates();
    }
}
```

আমরা জানি যে জাভা প্রোগ্রাম চালু করতে হলে একটি মেইন মেথড দরকার হয়। উপরের প্রোগ্রামটিতে একটি মেইন মেথড আছে। এবং এর ভেতরে শুরুতে আমরা দুইটি অবজেক্ট তৈরি করেছি।

```
Bicycle bike1 = new Bicycle();
Bicycle bike2 = new Bicycle();
```

জাভাতে অবজক্ট তৈরি করা খুব সহজ। এর জন্যে আমাদের তিনটি স্টেপ দরকার হয়-

- ডিক্লারেশন
- ইনসটেনশিয়েশান
- ইনিশিয়ালাইজেশান

Bicycle bike1 = new Bicycle();

উপরের বোল্ড অক্ষরে লেখাটুকু হচ্ছে ডিক্লারেশন, তারপর সমান চিহ্ন এর পর new কিওয়ার্ড পর্যন্ত হচ্ছে ইনসটেনশিয়েশান এবং এর পরের অংশটুকুকে ইনিশিয়ালাইজেশান বলা হয়। ইনিশিয়ালাইজেশান এর জন্যে আমাদের ক্লাসটির কনস্ট্রাকটরকে কল করতে হয়। কনস্ট্রাকটর নিয়ে একটু পরেই কথা বলছি।

এখানে ডিক্লারেশন টাইপ ডিক্লারেশন এর মতোই। ভ্যারিয়বল চ্যাপ্টারে আমরা আরো ডিটেইলস দেখবো।

তারপর অবজেক্টটি ধরে ডট অপারেটর ব্যবহার করে সেই ক্লাসের মেথড গুলো কল করা হয়ছে। এই প্রোগ্রামটি রান করলে আউটপুট আসবে-

```
cadence:50 speed:10 gear:2
cadence:40 speed:20 gear:3
```

কনস্ট্রাকটর

কনস্ট্রাকটর অন্যান্য মেথড বা ফাংশনের মতই একটি মেথড বা ফাংশনে। তবে এটির কোন রিটার্ন টাইপ নেই। একটি ক্লাসকে একটি অবজেক্ট-এ তৈরি করতে যে প্রয়োজনীয় কাজ গুলো করতে হয়, কনস্ট্রাকটর সেই কাজ গুলো করে থাকে। তবে মজার ব্যপার হচ্ছে সেই প্রয়োজনীয় কাজ গুলো জন্যে আমাদের কোড লিখতে হয় না।

আমাদের উপরের ক্লাসটিতে আমরা কোন কনস্ট্রাকটর লিখি নি। তাহলে এর অবজেক্ট তৈরি হলো কিভাবে? উত্তরটি হচ্ছে আমরা যদি কোন কনস্ট্রাকটর না লিখি তাহলে জাভা কম্পাইলার নিজে থেকেই একটি কনস্ট্রাকটর লিখে কম্পাইল করে, যাকে আমরা বলি ডিফল্ট কনস্ট্রাকটর। তবে আমরা চাইলে নিজের একটি লিখতে পারি।

```
public class Bicycle {
    int cadence = 0;
    int speed = 0;
    int gear = 1;

    public Bicycle() {
    }
}
```

এবার আমরা দেখবো কিভাবে জাভাতে কমেন্ট লিখতে হয়-

জাভা তিন ধরণের কমেন্ট সাপোর্ট করে-

```
Comment                                                 |  Description
    /* text */                                          |   জাভা কম্পাইলার /* থেকে */ পর্যন্ত সব কিছু উপেক্ষা
// text                                                  |   জাভা কম্পাইলার //   থেকে লাইনের শেষ পর্যন্ত উপেক্ষা
    /** documentation */                    | এটি হচ্ছে ডকুমেন্টাশান কমেন্ট। একে  doc comment বলা হয়
```

উদাহরণ-

```
package bd.com.howtocode.java.tutotorial.syntax;

/**
 * The HelloWorld program implements an application that
 * simply displays "Hello World!" to the standard output.
 *
 * @author Bazlur Rahman Rokon
 * @since 9/20/14.
 */
public class HelloWorld {
    public static void main(String[] args) {
        // Prints Hello, World! on standard output.
        System.out.println("Hello, world!");
```

```
            /*
            for (int i = 0; i < 100; i++) {
                System.out.println(i);
            }*/
        }
    }
```

আরও কিছু নিয়ম:

- জাভাতে প্রত্যেকটি স্টেটমেন্ট এর পর সেমিকোলন (;) দিয়ে স্টেটমেন্ট শেষ করতে হয়।
- জাভা একটি কেইস সেনসিটিভ ল্যাংগুয়েজ- অর্থাৎ hello এবং Hello দুটি আলাদা শব্দ ।

অনুশীলন:

নিচের প্যাটার্নগুলো প্রিন্ট করতে চেষ্টা করুন -

```
* * * * *          * * * * *    * * * * *
* * * * *        *       *     *     *
* * * * *          *       *      *   *
* * * * *        *       *        * *
* * * * *          * * * * *        *
   (a)            (b)          (c)
```

পাঠ ৩: ডাটা টাইপস এবং অপারেটর

- ভেরিয়েবল
- প্রিমিটিভ ডাটাটাইপ, ইন্টিজার, লং, ডাবল, ইন্টিজার, ফ্লোট এবং কার
- রেপার ক্লাস
- লিটারেল
- বিভিন্ন রকম অপারেটর

ভ্যারিয়বল

ভ্যারিয়বল হচ্ছে একটি নাম যা কম্পিউটারের একটি মেমোরি লোকেশান কে নির্দেশ করে। উদাহরণ-

```
int cadence = 0;
```

একটি ভ্যারিয়বল ডিক্লারেশন এর জন্যে একটি ডাটাটাইপ দরকার হয়, অর্থাৎ ভ্যারিয়বল টি কি ধরণের ডাটা হোল্ড করতে তা বলে দিতে হবে। উপরের উদাহরণটিতে আমরা একটিভ্যারিয়বল ডিক্লার করেছি যার নাম cadence এবং এটি ইন্টিজার টাইপ ডাটা হোল্ড করে।

যেহেতু জাভা একটি স্ট্যাটিক্যালি টাইপড ল্যাংগুয়েজ সুতরাং ভ্যারিয়বল ডিক্লারেশন এর সময় ডাটাটাইপ উ্লেখ করা অত্যাবশ্যক।

জাভাতে আমরা চার ধরণের ভেরিয়েবল নিয়ে কাজ করে থাকি -

1. Instance Variables (Non-static fields)
2. Class Variables (Static Fields)
3. Local variables
4. Parameters variables

জাভাতে ভ্যারিয়বল এবং ফিল্ড দুই শব্দই ব্যবহার করা হয়, তবে এর কিছু টেকনিকাল পার্থক্য আছে। সেগুলো নিয়েই আলোচনা করা হবে –

আমরা আবার একটি উদাহরণ দেখি –

```
public class Bicycle {
    static int numGears = 6;

    int cadence = 0;
    int speed = 0;
    int gear = 1;

    public Bicycle() {
    }

    void changeCadence(int newValue) {
        cadence = newValue;
    }

    void changeGear(int newValue) {
        gear = newValue;
    }

    void speedUp(int increment) {
        speed = speed + increment;
    }

    void applyBrakes(int decrement) {
        speed = speed - decrement;
    }

    void printStates() {
        System.out.println("cadence:" +
                cadence + " speed:" +
                speed + " gear:" + gear);
    }
}
```

আমরা জানি যে একটি ক্লাস থেকে আমরা অবজেক্ট তৈরি করি। আমরা একটা ক্লাস থেকে অনেকগুলো অবজেক্ট তৈরি করতে পারি। এবং প্রত্যেক অবজেক্ট-ই আলাদা আলাদা। যেমন -

```
Bicycle bike1 = new Bicycle();
Bicycle bike2 = new Bicycle();
```

এখানে bike1 এবং bike2 দুটি সম্পূর্ণ আলাদা অবজেক্ট।

এখন bike1 এবং bike2 তে কিছু ভ্যারিয়বল গুলোও আলাদা। অর্থাৎ আমরা যতগুলো অজেক্ট তৈরি করতেবা ঠিক ততোগুলো আলাদা ভ্যারিয়বল থাকবে মেমোরিতে। এক্ষেত্রে মেমোরিতে ২টা cadence থাকবে, ২টা gear থাকবে এবং ২ speed ধাকবে।

এই ভ্যারিয়বল গুলোকে Instance Variables বা Non-static fields বলা হয়। এই ভ্যারিয়বল গুলো আগে static কিওয়ার্ডটি থাকে না।

```
static int numGears = 6;
```

উপরের উদাহরণটিতে **numGears** নামে একটি ভ্যারিয়বল আছে, এটির আগে একটি **static** কিওয়ার্ডটি আছে। এ ধরণের ভ্যারিয়বল কে Class Variables বা Static Fields বলা হয়। static কিওয়ার্ডটি কম্পাইলারকে বলে যে numGears নামে একটি মাত্র ভ্যারিয়বল থাকবে মেমোরিতে, অবজেক্ট এর সংখ্যা যতই হোক।

লোকাল ভ্যারিয়বল হলো সেসব ভ্যারিয়বল যে গুলো কোন মেথডের মাঝে ডিক্লার করা হয়। একটি লোকাল ভ্যারিয়বল শুধু মাত্র সেই মেথডের ভেতর থেকেই একসেস করা যাবে।

আর Parameters variables হলো সেই ভ্যারিয়বল গুলো যেগুলো মেথড কল করার সময় পাস করা হয়। এ গুলোও শুধুমাত্র মেথডের ভেতর থেকেই একসেস করা যায়।

আমরা Instance Variables এবং Class Variables গুলোকে ফিল্ড বলি।

এখানে কিছু ভ্যারিয়বল ডিক্লারাশেনের উদাহরণ দেওয়া হলো -

```
byte    myByte;
short   myShort;
char    myChar;
int     myInt;
long    myLong;
float   myFloat;
double  myDouble;
```

শুরুতে আগে টাইপ লিখতে হবে, তারপর একটি নাম, তারপর সেমিকোলন দিয়ে শেষ করতে হবে। তবে আমরা চাইলে ভ্যারিয়বল কে ইনিশিয়ালাইজেশান করতে পারি। যেমন -

```
     int cadence = 0;
```

অর্থাৎ শুরুতে আমরা cadence এর ভ্যালু 0 এসাইন করালাম।

এরপর যদি আমরা কোন ভ্যারিয়বলে ভ্যালু এসাইন করতে চাই তাহলে -

```
myByte   = 127;
myFloat  = 199.99;
```

জাভা ভ্যারিয়বল লেখার কিছু নিয়ম কানুন আছে-

1. ভ্যারিয়বল গুলো কেইস সেনসিটিভ। অর্থাৎ money, Money,MONEY তিনটি আলাদা।
2. ভ্যারিয়বল অবশ্যই যেকোন একটি লেটার দিয়ে শুরুকরতে হবে। তবে $ অথবা _ দিয়েও শুরু করা যায়।
3. ভ্যারিয়বল এর মাঝে নাম্বার কিংবা _ থাকতে পারে।

4. ভ্যারিয়বল জাভার কোন reserved কিওয়ার্ড হতে পারবে না।

ডাটা টাইপ

জাভা তে আট ধরণের প্রিমিটিভ ডাটা টাইপ আছে ।

Data type	Description
byte	8 bit signed value, values from -128 to 127
short	16 bit signed value, values from -32.768 to 32.767
char	16 bit Unicode character
int	32 bit signed value, values from -2.147.483.648 to 2.147.483.647
long	64 bit signed value, values from -9.223.372.036.854.775.808 to 9.223.372.036.854.775.808
float	32 bit floating point value
double	64 bit floating point value

এগুলো প্রমিটিভ , এর মানে হচ্ছে এগুলো অবজেক্ট নয়। এরা মেমোরিতে সরাসরি ভ্যালু রাখে।

রেপার ক্লাস

তবে জাভাতে কিছু ডাটা টাইপ আছে যেগুলো অবজেক্ট।

Data type	Description
Byte	8 bit signed value, values from -128 to 127
Short	16 bit signed value, values from -32.768 to 32.767
Character	16 bit Unicode character
Integer	32 bit signed value, values from -2.147.483.648 to 2.147.483.647
Long	64 bit signed value, values from -9.223.372.036.854.775.808 to 9.223.372.036.854.775.808
Float	32 bit floating point value
Double	64 bit floating point value

এগুলোকে প্রিমিটিভ টাইপ এর রেপার রেপার ক্লাস বলা হয়। লক্ষ্য করুণ, এগুলোর সবগুলোর নাম ক্যাপিটাল অক্ষর দিয়ে শুরু হয়েছে।

তবে আমরা চাইলে অবজেক্ট ডাটাটাইপ এবং প্রিমটিভ ডাটাটাইপ একে অপরের পরিপূরক হিসাবে ব্যবহার করতে পারি।

```
Integer a;
int b = 9;
a = b;
```

তবে প্রিমিটিভ ভ্যালু গুলো ডিফল্ট ভ্যালু থাকে। অর্থাৎ আমরা যদি ভ্যালু এসাইন না করি, তাহলে এদের মধ্যে বাইডিফল্ট ভ্যালু থাকে। যেমন -

Data Type	Default Value (for fields)
byte	0
short	0
int	0
long	0L
float	0.0f
double	0.0d

char	'\u0000'
boolean	false

লিটারেল-

প্রোগ্রামিং ল্যাংগুয়েজ কিছু মজার মজার বিল্ট-ইন সুবিধা থাকে, তার মধ্যে লিটারেল একটি। আমরা জানি যে একটা ভ্যারিয়বল ডিক্লারেশান এর জন্য প্রথমে টাইপ লিখতে হয়, তারপর একটা নাম দিতে হয়, তারপর একে ইনিশিয়ালাইজেশান করতে হয়। ভেরিয়েবলটি যদি অবজেক্ট হয়, তাহলে ইনটেনশিয়েশান করতে হয়।

উদাহরণ-

```
List list = new ArrayList();

or

Int x = 5;
```

উপরের দুটি উদাহরণের মাঝে একটিতে আমরা new কিওয়ার্ড ব্যবহার করে নতুন অবজেক্ট তৈরি করেছি। কিন্তু পরের উদাহরণটিতে সেটি করতে হয় নি। আমরা সরাসরি একটি ভ্যালু এসাইন করেছি। এখানে 5 একটি ভ্যালু। এখানে 5 হচ্ছে লিটারেল।

এরকম অনেক ক্ষেত্রে আমরা new কিওয়ার্ড ব্যবহার না করেই ভেরিয়েবল initialize করতে পারি।

জাভাতে প্রিমিটিভ টাইপ সকল ডাটাটাইপ লিটারেল সাপোর্ট করে। যেমন -

```
boolean result = true;
char capitalC = 'C';
byte b = 100;
short s = 10000;
int i = 100000;
```

নিচে আরো কিছু উদাহরণ দেওয়া হলো –

ইন্টিজার লিটারেল-

```
  // এখানে 26 হচ্ছে ডেসিমাল নাম্বার
  int decVal = 26;
  //  এখানে 26 সংখ্যাটি হেক্সাডেসিমেল এ দেখানো হয়েছে
  int hexVal = 0x1a;
  //  এখানে 26 সংখ্যাটি বাইনারি-তে এ দেখানো হয়েছে
  int binVal = 0b11010;
```

ফ্লোটিং পয়েন্ট লিটারেল-

```
 double d1 = 123.4;
// একি ভ্যালু বৈজ্ঞানিক উপায়ে লেখা হয়েছে
double d2 = 1.234e2;
float f1  = 123.4f;
ক্যারেক্টার এন্ড স্ট্রিং লিটারেল--
```

char এবং String উদ্ধৃতি চিহ্নের ভেতরে লেখা হয়। char ক্ষেত্রে একক উদ্ধৃতি চিহ্ন String এর জন্যে ডবল উদ্ধৃতি চিহ্ন ব্যবহার করতে হয়- যেমন-

```
char chr = 'A'; // ক্যারেক্টার লিটারেল
String name = "Bazlur"; // স্ট্রিং লিটারের
```

char এবং String ইউনিকোড ক্যারেক্টার হতে পারে।

আমরা জানি কিভাবে ভেরিয়েবল ইনিশিয়ালাইজ করতে হয় জানি, এবার তাহলে এই ভ্যারিয়বল গুলো দিয়ে কি কাজ করা যায় সেগুলো দেখি।

কোন কাজ করতে হলে একজন কার্যকারী বা অপারেটর লাগে। অপারেটর কিছু অপারেন্ড নিয়ে কাজ করে থাকে তারপর ফলাফল রিটার্ন করে। জাভা প্রোগ্রামিং ল্যাংগুয়েজ এ বেশ কিছু অপারেটর আছে- সেগুলো দেখা যাক-

এসাইনমেন্ট অপারেটর **(Assignment Operator)**

"=" এটি হচ্ছে এসাইনমেন্ট অপারেটর বাংলায় যাকে বলে সমান সমান চিহ্ন। আমরা একটি Bicycle ক্লাস দেখেছি, এর মাঝে কিছু ভেরিয়েবল দেখেছি-

```
 int cadence = 0;
int speed = 0;
int gear = 1;
```

এই ভ্যারিয়বল গুলোর ডান পাশে সমান সমান চিহ্নের পর আমরা একটা ভ্যালু বা মান বসিয়েছি। এভাবে আমরা একটি ভ্যারিয়বল এর মাঝে ভ্যালু এসাইন করতে পারি।

এরিথমেটিক অপারেটর**(Arithmetic Operator)**

জাভা প্রোগ্রামিং ল্যাংগুয়েজ-এ যোগ, বিয়োগ, গুন, ভাগ করার জন্যে কিছু অপারেটর আছে। এগুলো আমরা যখন বেসিক গণিত শিখি তখন থেকেই জানি। শুধু একটি অপারেটর নতুন মনে হতে পারে, যা হলো "%"। এটিকে অনেকেই পারসেন্টেজ বা শতকরা চিহ্ন হিসেবে ভুল করতে পারে, কিন্তু এটি আসলে তা নয় । এটি মূলত একটি সংখ্যাকে আরেকটি সংখ্যা দ্বারা ভাগ করে ভাগশেষ রিটার্ন করে।

অপারেটর	এর কাজ
+	আডিটিভ(Additive) অপারেটর, যা দুটি সংখ্যা বা স্ট্রিং যোগ করার জন্যে ব্যবহার করা হয়।
-	সাবস্ট্রাকশান (Subtraction) অপরেটর যা একটি সংখ্যা থেকে আরেকটি সংখ্যা বিয়োগ করার জন্যে ব্যবহার করা হয়।
*	মাল্টিপ্লিকেশান (Multiplication)অপারেটর যা দুটি সংখ্যাকে গুন করে।
/	ডিভিশান(Division) অপারেটর , যা দিয়ে একটি সংখ্যাকে আরেকটি সংখ্যাকে ভাগ করা যায়।
%	রিমাইন্ডার (Remainder) অপারেটর যা একটি সংখ্যাকে আরেকটি সংখ্যা দ্বারা ভাগ করে ভাগশেষ রিটার্ন করে।

```
class ArithmeticDemo {

    public static void main (String[] args) {

        int result = 1 + 2;
        // এখানে result এর মান হচ্ছে 3
        System.out.println("1 + 2 = " + result);
        int original_result = result;

        result = result - 1;
        //এখানে result থেকে ১ সাবস্ট্রাক্ট করায় এর মান ২
        System.out.println(original_result + " - 1 = " + result);
        original_result = result;

        result = result * 2;
        // এখানে result এর সাথে ২ মাল্টিপ্লাই করার ফলে এর মান 4
        System.out.println(original_result + " * 2 = " + result);
        original_result = result;

        result = result / 2;
        //আবার result ডিভাইড করার ফলে এর মান হয়ে গেল 2
        System.out.println(original_result + " / 2 = " + result);
        original_result = result;

        result = result + 8;
        // ৮ যোগ করার ফলে এর result হলো  10
        System.out.println(original_result + " + 8 = " + result);
        original_result = result;

        result = result % 7;
        // result এর সাথে ৭ রিমাইন্ডার অপারেটর ব্যবহার করার ফলে এর মান হয়ে গেল 3, কারণ এটি শুধু মাত্র রিমানইন্ডার বা ভাগশেষ রিটার্ন ক
        System.out.println(original_result + " % 7 = " + result);
    }
}
```

এই প্রোগ্রামটি রান করলে নিচের ফলাফল প্রকাশিত হবে।

```
1 + 2 = 3
3 - 1 = 2
2 * 2 = 4
4 / 2 = 2
2 + 8 = 10
10 % 7 = 3
```

ইউনারি **(Unary)** অপারেটর

উপরের সব অপারেটর এর জন্যে আমাদের দুটি করে অপারেন্ড দরকার হতো, তবে এই অপারেটর এর লাগে একটি।

এগুলো বিভিন্ন ধরণের কাজ করে থাকে যেমন – এক করে ইনক্রিমেন্টিং/ডিক্রিমেন্টিং বা একটা এক্সপ্রেশান নেগেট করা বা একটা বুলিয়ান-কে ইনভার্ট করা । এগুলো হল - +, -, ++, —, ! উহাহরণ -

```
class UnaryDemo {

  public static void main(String[] args) {

      int result = +1;
      // এটি এক করে ইনক্রিমেন্ট করে,  সুতরাং এখানে  result এর মান 1
      System.out.println(result);

      result--;
      // এটি এক করে ডিক্রিমেন্ট করে, সুতরাং  এখানে  result এর মান 0
      System.out.println(result);

      result++;
      // এটিও এক করে ইনক্রিমেন্ট করে, সুতরাং  এখানে  result এর মান আবার ১
      System.out.println(result);

      result = -result;
     // এখানে     result কে নেগেট করে, সুতরাং এর মান এখন -1
      System.out.println(result);

      boolean success = false;
      // এখানে বুলিয়ানের মান হচ্ছে  false
      System.out.println(success);
      // কিন্তু এর আগে একটি নেগেট অপারেটর এড করলে এটি হয়ে যায়
      System.out.println(!success);
  }
}
```

ইকুয়ালিটি **(Equality)** এবং রেশনাল**(Relational)** অপারেটরস

ইকুয়ালিটি (Equality) এবং রেশনাল(Relational) অপারেটর গুলো নির্ধারণ করে একটি ভ্যালু অন্যটি থেকে বড় বা ছোট কিনা।

```
==      দুটি ভ্যালু সমান হলে এই এক্সপ্রেশান এর মান true হয়
!=       দুটি ভ্যালু সমান না হলে  true হয়
>       প্রথম ভ্যালু পরের ভ্যালু থেকে বড় হলে true হয়
>=      প্রথম ভ্যালু পরের ভ্যালু থেকে বড় বা সমান হলে true হয়
<      প্রথম ভ্যালু পরের ভ্যালু থেকে ছোট হলে true হয়
<=     প্রথম ভ্যালু পরের ভ্যালু থেকে ছোট বা সমান হলে true হয়
```

উদাহরণ

```
class ComparisonDemo {

    public static void main(String[] args){
        int value1 = 1;
        int value2 = 2;
        if(value1 == value2)
            System.out.println("value1 == value2");
        if(value1 != value2)
            System.out.println("value1 != value2");
```

```
            if(value1 > value2)
                System.out.println("value1 > value2");
            if(value1 < value2)
                System.out.println("value1 < value2");
            if(value1 <= value2)
                System.out.println("value1 <= value2");
        }
    }
```

কন্ডিশনাল(Conditional) অপারেটর

&& এবং || এই দু:ই অপারেটরকে কন্ডিশনাল অপারেটর বলে।

```
&&            কন্ডিশনাল অ্যান্ড (Conditional-AND)
 ||                 কন্ডিশনাল অর  ( Conditional-OR)
```

উদাহরণ-

```
    class ConditionalDemo1 {

       public static void main(String[] args){
           int value1 = 1;
           int value2 = 2;
           if((value1 == 1) && (value2 == 2))
               System.out.println("value1 is 1 AND value2 is 2");
           if((value1 == 1) || (value2 == 1))
               System.out.println("value1 is 1 OR value2 is 1");
       }
    }
```

চলবে --

এরে (Array)

এরে

এরে হচ্ছে একধরণের কন্টেইনার অবজেক্ট যা অনেকগুলো একিধরণের ডাটা টাইপের এর একটি ফিক্সড সাইজের ভ্যালু ধরে রাখতে পারে।

এরে ডিক্লার করার জন্যে প্রথমে ডাটাটাইপ (কি ধরণের ডাটাটাইপ রাখবে) এর সাথে ([]) স্কয়ার ব্র্যাকেট তারপর এর একটি ভেরিয়েবল নাম দিতে হয়।

```
    //একটি ইন্টিজার এরে
    int[] anArray;
```

তবে স্কয়ার ব্র্যাকেট ভেরিয়েবল নাম এর পরেও দেওয়া যেতে পারে - উদহরণ-

```
        int anArray[];
```

এভাবে আমরা অন্য ডাটাটাইপ এর অ্যারে লিখতে পারি -

```
long[] anArrayOfLongs;
float[] anArrayOfFloats;
double[] anArrayOfDoubles;
```

এরে একটি অবজেক্ট, সুতরাং একে নিউ(new) অপারেটর দিয়ে প্রথমে ক্রিয়েকট করতে হবে।

```
// এখানে ১০ সাইজের একটি এরে ক্রিয়েট করা হলো
anArray = new int[10];
```

এই স্ট্যাটমেন্ট যদি না লেখা হয় তাহলে প্রোগ্রামটি কম্পাইল হবে না।

এরপর আমরা এর এর ভেতর ভ্যাল্যু রাখতে পারি।

```
anArray[0] = 100; //এখানে প্রথম ভ্যাল্যু রাখা হল
anArray[1] = 200; // এভাবে দ্বিতীয় ভ্যালু
anArray[2] = 300; // এভাবে বাকি গুলো
```

এই ভ্যালুগুলো যদি পড়তে চাই তাহলে -

```
System.out.println("Element 1 at index 0: " + anArray[0]);
System.out.println("Element 2 at index 1: " + anArray[1]);
System.out.println("Element 3 at index 2: " + anArray[2]);
```

এছাড়াও এরে লেখার শর্টকাট পদ্ধতি আছে -

```
int[] anArray = {
    100, 200, 300,
    400, 500, 600,
    700, 800, 900, 1000
};
```

একটি এরে এর মধ্যে আরেকটি অ্যারে রাখা যেতে পারে -

```
int[][] anArray = {{1, 2, 3}, {4, 6, 7}, {8, 9}};
```

আমরা যদি একটি এরে এর লেন্থ বা সাইজ জানতে চাই তাহলে –

```
int length = anArray.length;
```

এক্সপ্রেশান(Expressions), স্টেটমেন্ট(Statements) এবং ব্লক(Blocks)

আমরা ইতিমধ্যে ভেরিয়েবল এবং অপারেটর সম্পর্কে জেনে ফেলেছি, এবার তাহলে আমরা জেনে নিই এক্সপ্রেশান কি।

এক্সপ্রেশান**(Expressions)**

একপ্রেশান হচ্ছে কতগুলো ভ্যারিয়েবল, অপারেটর এবং মেথড বা ফাংশান কল এর মাধ্যমে একটি আউটপুট তৈরি করার জন্যে যে কোড লেখা হয়। উদাহরণ-

```
int cadence = 0;
anArray[0] = 100;

int result = 1 + 2;
if (value1 == value2)
    System.out.println("value1 == value2");
```

উপরের cadence = 0 একটি এক্সপ্রেশান। এটির "=" অপারেটরের মাধ্যমে একটি ভ্যালু cadence ভ্যারিয়েবল এ এসাইন হয়। তারপর anArray[0] = 100 এই এক্সপ্রেশানের মাধ্যমে anArray এরে এর প্রথম ঘরে 100 এসাইন করা হল।

1 + 2 একটি এক্সপ্রেশান যা "+" অপারেটর এর মাধ্যমে দুটি সংখ্যা যোগ হয় এবং "=" অপারেটর এর মাধ্যমে result ভ্যারিয়েবল এ এসাইন হয়। সুতরাং এখানে দুইটা এক্সপ্রেশান।

জাভা প্রোগ্রামিং ল্যাংগুয়েজ কম্পাউন্ড এক্সপ্রেশান সাপোর্ট করে। এর মানে হচ্ছে অনেকগুলো ছোট ছোট এক্সপ্রেশান নিয়ে আমরা একটি বড় এক্সপ্রেশান তৈরি করতে পারি। একটি এক্সপ্রেশান মূলত একটি নির্দিষ্ট ডাটাটাইপ এর ভ্যালু প্রদান করে, সুতরাং কম্পাউন্ড এক্সপ্রেশান এর ক্ষেত্রে সব এক্সপ্রেশান এর ফলাফল একি ডাটাটাইপ এর হতে হবে।

```
1 * 2 * 3
```

এখানে 1 * 2 একটি এক্সপ্রেশান যার ইন্টিজার টাইপ এর ডাটাটাইপ এর আউটপুট প্রদান করে, এবং এটি যখন আবার 3 এর মাল্টিপ্লাই করা হয়, তখনও এর আউটপুট ইন্টিজার টাইপ হয়।

তবে কম্পাউন্ড এক্সপ্রেশান এর ক্ষেত্রে এম্বিগিউটি দূর করার জন্যে ব্রেস "()" ব্যবহার করা উত্তম। উদাহরণ -

```
x + y / 100
```

এবং (x + y) / 100

এই দুটি এক্সপ্রেশান এর ফলাফল ভিন্ন হবে।

তবে যদি এক্সপ্রেশান এর অর্ডার ব্রেস দিয়ে না ঠিক করে দেওয়া হয় তবে অপারেটর এর অগ্রগণ্যতা(precedence) অনুযায়ী এক্সপ্রেশান এর অর্ডার নির্ধারিত হয়।

স্টেটমেন্টস**(Statements)**

স্টেটমেন্টস হচ্ছে অনেকটা একটা পূর্ণাঙ্গ বা সার্থক বাংলা বাক্যেরে মতো। তবে প্রোগ্রামিং এর ভাষায় এটি হচ্ছে- একটি ছোট ইউনিট অব কোড যা কিনা এক্সিকিউশান করা যায়। কতগুলো এক্সপ্রেশান শেষে সেমিকোলন (;) দিয়ে শেষ করলে স্টেটমেন্ট হয়ে যায়। যেমন-

- এসাইনমেন্ট এক্সপ্রেশান
- ++ অথবা-- এর ব্যবহার
- মেথড/ফাংশান কল
- নতুন অবজেক্ট তৈরি করা, ইত্যাদি ।

এদেরকে এক্সপ্রেশানাল স্টেটমেন্ট বলা হয়।

```
// এটি এ্যাসাইনমেন্ট স্টেটমেন্ট
aValue = 8933.234;
// এটি ইনক্রিমেন্ট স্টেটমেন্ট
aValue++;
// এখানে একটি মেথড কল করা হয়েছে
System.out.println("Hello World!");
// এখানে একটি অবজেক্ট তৈরি করা হয়েছে
Bicycle myBike = new Bicycle();
```

আরও দু-ধরণের স্টেটমেন্ট আছে- ডিক্লারেশান স্টেটমেন্ট –

```
        double aValue = 8933.234;
```

কন্ট্রোল ফ্লো স্টেটমেন্ট – এটি নিয়ে পরবর্তী চ্যাপ্টারে আরও বিস্তারিত বলা হবে।

ব্লকস(Blocks)

একটি কার্লি ব্রেস "{}" এর মাঝে শূণ্য অথবা একাধিক স্টেটমেন্ট থাকলে তাকে ব্লক বলা হয়। উদারহরণ-

```
class BlockDemo {
     public static void main(String[] args) {
          boolean condition = true;
          if (condition) { //এখানে ব্লক -১ এর শুরু
               System.out.println("Condition is true.");
          } //  এখানে ব্লক -১ শেষ
          else { // এখানে ব্লক-২ শুরু
               System.out.println("Condition is false.");
          } // এখানে ব্লক-২ শেষ
     }
}
```

পাঠ ৪: কন্ট্রোল ফ্লো -লুপিং- ব্রাঞ্চিং

- ইফ-দেন-ইলস
- সুইচ
- ফর লুপস
- হুয়াইল লুপ
- ডু-ইয়াইল লুপ
- ব্রেক স্ট্যাটমেন্ট
- কন্টিনিউ স্ট্যাটমেন্ট
- রিটার্ন স্ট্যাটমেন্ট
- সারসংক্ষেপ

আমাদের সোর্সকোডে -এ যেসব স্টেটমেন্ট থাকে তা সাধরারণত উপর থেকে নিচে যে অর্ডার এ দেওয়া থাকে সেই অর্ডারেই এক্সিকিউট হয়। কিন্তু কন্ট্রোল ফ্লো স্টেটমেন্ট এই অর্ডারকে ভেঙ্গে বিভিন্ন ডিসিশান মেকিং, লুপিং এবং ব্রাঞ্চিং এর মাধ্যমে একটি নির্দিষ্ট কোড ব্লক-কে এক্সিকিউট করে।

কন্ট্রোল ফ্লো স্টেটমেন্ট গুলি হচ্ছে -

- ডিসিশান-মেকিং স্টেটমেন্ট (if-then, if-then-else, switch)-
- লুপিং স্টেটমেন্ট (for, while, do-while)
- এবং ব্রাঞ্চিং স্টেটমেন্ট (break, continue, return)

`if-then` স্টেটমেন্ট হচ্ছে সব চেয়ে বেসিক কন্ট্রোল ফ্লো স্টেটমেন্ট।

আমরা যদি একটি প্রোগ্রাম এর একটি নির্দিষ্ট কোড ব্লক শুধু মাত্র একটি বিশেষ কন্ডিশান বা শর্ত সাপেক্ষে এক্সিকিউট করতে চাই তাহলে আমরা `if-then` স্টেটমেন্ট ব্যবহার করি-

উদাহরণ-

```
int x = 10;

  if( x < 20 ){
     System.out.print("This is if statement");
  }
```

উপরের কোড ব্লকটিতে আমরা শুধু মাত্র x এর মান 20 হলেই তা প্রিন্ট করতে চাই।

`if` স্টেটমেন্ট এর পেরেনস্থেসিস "()" মাঝে একটি বুলিয়ান এক্সপ্রেশান থাকে। বুলিয়ান এক্সপ্রেশান হচ্ছে এক ধরণের এক্সপ্রেশান যার ফলাফল শুধুমাত্র `true` অথবা `false` হতে পারে। এই বুলিয়ান এক্সপ্রেশানটির মান যদি `true` হয় তাহলে এই if স্টেটমেন্ট এর ব্লকটি এক্সিকিউট হবে, নতুবা হবে না।

তবে আমাদের প্রথম কন্ডিশান বা শর্ত বা বুলিয়ান এক্সপ্রেশান যদি সত্যি না হয়, এবং এক্ষেত্রে আমরা অন্য একটি ব্লক অব কোড এক্সিকিউট করতে চাই, তাহলে `if-then-else` স্টেটমেন্ট ব্যবহার করি। উদাহরণ-

```
  if( x < 20 ){
     System.out.print("This is if statement");
  }else{
     System.out.print("This is else statement");
  }
```

উপরের উদাহরণটি-তে একটি কন্ডিশান বা বুলিয়ান এক্সপ্রেশান ছিল, কিন্তু আমাদের মাঝে মাঝে একাধিক কন্ডিশান থাকতে পারে। তাহলে আরেকটি উদাহরণ দেখা যাক-

```
    int score = 76;
    char grade;

    if (score >= 90) {
        grade = 'A';
    } else if (score >= 80) {
```

```
            grade = 'B';
        } else if (score >= 70) {
            grade = 'C';
        } else if (score >= 60) {
            grade = 'D';
        } else {
            grade = 'F';
        }

        System.out.println("Grade = " + grade);
```

উপরের উদাহরণটি যদি আমরা রান করি তাহলে output হবে -

```
Grade = C
```

এখানে প্রথম বুলিয়ান এক্সপ্রেশানটি যদি `true` হয়, তাহলে `grade = 'A';` কোড ব্লকটি এক্সিকিউট হবে, আর `true` না হয়, তাহলে পরের কোড ব্লক, অর্থাৎ `else if (score >= 80)` এক্সপ্রেশানটি ইভালুয়েট করা হবে, এবং এটি যদি `true` হয় তাহলে এর কার্লি ব্রেস `{}` এর মাঝের কোড ব্লকটি এক্সিকিউট হবে। অর্থাৎ আমাদের যদি অনেকগুলো কন্ডিশান থাকে তাহলে আমরা `if` কন্ডিশান এর সাথে `else if` দিয়ে সেগুলো-কে এড করতে পারি। এই কন্ডিশান গুলোর মধ্যে যে কোন একটি এক্সপ্রেশান যদি `true` হয় তাহলে সেই ব্লক এর কোডটি এক্সিকিউট হবে ।

এখানে লক্ষ্য রাখতে হবে যে, প্রথম এক্সপ্রেশানটি যদি `true` হয়, তাহলে কিন্তু বাকি কন্ডিশান গুলো আর ইভালুয়েট হবে না। অর্থাৎ রান টাইমে এই কোড ব্লক গুলো একদম প্রথম `if` কন্ডিশান থেকে যতক্ষন পর্যন্ত কোন `true` এক্সপ্রেশান না পাওয়া যায়, ঠিক ততক্ষণ পর্যন্ত এক্সপ্রেশন গুলো ইভালুয়েট হবে। আমাদের উদাহরণটিতে - প্রথম, দ্বিতীয় এবং তৃতীয় এই তিনটি এক্সপ্রেশান ইভালুয়েটেড হয়েছে, এবং তৃতীয়টিতে `true` এক্সপ্রেশান পাওয়া গেছে, এবং `grade = 'C';` এই কোড ব্লকটি এক্সিকিউট হয়েছে।

এভাবে আমাদের যদি একাধিক কন্ডিশান এর জন্য আমরা `if-then-else` ব্যবহার করে কোড লিখতে পারি। যদি একাধিক

Switch

আমাদের কোড এ যদি একাধিক এক্সিকিউশান পাথ থাকে তাহলে, আমরা `if-then` এবং `if-then-else` ব্যবহার করে কোড লিখতে পারি। তবে এর পরিবর্তে `switch` স্ট্যাটমেন্ট ও ব্যবহার করতে পারি। উদাহরণ-

```
public static String getMonth(int month) {
        String monthString;

        switch (month) {
            case 1:
                monthString = "January";
                break;
            case 2:
                monthString = "February";
                break;
            case 3:
                monthString = "March";
                break;
            case 4:
                monthString = "April";
                break;
            case 5:
                monthString = "May";
                break;
            case 6:
                monthString = "June";
                break;
            case 7:
                monthString = "July";
                break;
            case 8:
                monthString = "August";
                break;
            case 9:
                monthString = "September";
                break;
            case 10:
                monthString = "October";
                break;
            case 11:
                monthString = "November";
```

```
                break;
            case 12:
                monthString = "December";
                break;
            default:
                monthString = "Invalid month";
                break;
        }
        return monthString;
    }
```

চলবে ----

পাঠ ৫: অবজেক্ট ওরিয়েন্টেড প্রোগ্রামিং-১

- অবজেক্ট
- ক্লাস
- ইন্টারফেইস
- অ্যাবস্ট্রাক্ট ক্লাস
- স্ট্যাটিক মেম্বার
- অবজেক্ট ওরিয়েন্টেড কনসেপ্ট
- ইনহেরিট্যান্স
- পলিমরফিজম
- ইনকেপসুলেশান
- সারসংক্ষেপ

শুরুতে বস্তুর ধারণা নিয়ে একটি ছোট্ট ব্যাখ্যা দেই, পরবর্তীতে আমি এই কথাগুলো আরো ব্যাখ্যা করে বলবো। আমরা সবাই কম্পিউটার ব্যবহার করি, যারা একটু বেশি কৌতূহলী তারা নিশ্চয় কম্পিউটারের বক্স খুলে খুলে দেখে ফেলেছে যে, এর মধ্যে নানা রকম যন্ত্রাংশ থাকে, যেমন, র‍্যাম, হার্ডডিস্ক, মাদারবোর্ড, সিপিইউ, কুলিং ফ্যান ইত্যাদি। এইসব মিলেই কম্পিউটার। কিন্তু মজার ব্যপার হলো, এর সবই একটি কোম্পানি তৈরি করেনি। কেও র‍্যাম তৈরি করে, কেওবা মাদারবোর্ড, কেও বা আবার সিপিইউ। কিন্তু সবাই আলাদা আলাদা ভাবে সবকিছু তৈরি করলেও আমরা যখন পুরো কম্পিউটারটি এসেম্বল করি, কি সুন্দর ভাবে সব ঠিক ঠাক ভাবে লেগে যায়, কোন সমস্যা হয় না। একজন ম্যাকানিক-ও যার নাকি কম্পিউটার সায়েন্স এ ডিগ্রি নেই, সেও জানে কিভাবে সব কিছু এসেম্বল করতে হয়। র‍্যাম এর মধ্যে কি আছে সে সম্পর্কে তার কোনই ধারণা নেই, কিংবা সিপিইউ । অবজেক্ট ওরিয়েন্টেড কনসেপ্ট মূল ব্যাপারটি হলো এটি। একটা সিস্টেমে অনেক গুলো কম্পোনেন্ট থাকতে পারে, কিন্তু সব কম্পোনেন্ট গুলো কেও একা তৈরি করবে না এইটাই স্বাভাবিক, এবং এগুলো এমন ভাবে তৈরি করা হয় যাতে খুব সহজেই এদেরকে এসেম্বল করে পুরো সিস্টেম দাড় করানো যায়।

অবজেক্ট ওরিয়েন্টেড কনসেপ্ট এর ধারণার সাথে পরিচিত হতে হলে শুরুতে আমাদের কিছু টার্ম বা শব্দের সাথে পরিচিত হতে হয়। আমি শুরুতে এনালজি বা উপমা দিয়ে বুঝানো চেষ্টা করবো, তারপর মূল বিষয়ে চলে আসবো।

অবজেক্ট **(Object):**

এর মানে হচ্ছে বস্তু। পৃথিবীতে যা কিছু দেখি, অনুভব করি, তার সবই বস্তু। যেমন- মোবাইল ফোন, চশমা, এমনকি আমি নিজেও একটি অবজেক্ট। আমরা যেহেতু প্রোগ্রামার, এখন একটু সেভাবে কথা বলি। প্রোগ্রামিং এ একটা ধারণাও অবজেক্ট। অবজেক্ট কে কিভাবে দেখা হচ্ছে তা নির্ভর করে যে দেখছে তার উপর। মনে করা যাক, একটি অফিসের বড়ো কর্তা (CEO) সে দেখবে, এমপ্লয়ি, বিল্ডিং, ডিভিশন, নোটপত্র, বেনিফিট প্যাকেজ, লাভ ক্ষতির হিসেব এগুলো অবজেক্ট। একজন আর্কিটেক্ট দেখবে, তার প্ল্যান, মডেল, এলেভেশান, ডোনেজ, ডেন্টিল, আর্মাচার ইত্যাদি। সেভাবে একজন সফটওয়্যার ইঞ্জিনিয়ারের অবজেক্ট হলো, স্ট্যাক, কিউ, উইন্ডো, চেক বক্স ইত্যাদি। অবজেক্ট এর একটি স্টেট থাকে। স্টেট হলো কিছু তথ্য যা দিয়ে ওই অবজেক্টকে আলাদা করা যায়, এবং তার বর্তমান অবস্থান জানা যায়। যেমন একটি ব্যাংক একাউন্ট স্টেট হতে পারে কারেন্ট ব্যালেন্স। একটা অবজেক্ট এর মধ্যে আরেকটি অবজেক্ট থাকতে পারে, যা ওই অবজেক্ট এর স্টেট হতে পারে।

অবজেক্ট সাধারণত কিছু কাজ করে থাকে যাকে বলে তার বিহেভিয়ার। যেমন ধরা যাক, সাইক্যালের চাকা, চাকার স্টেট হতে পারে এর ব্যাসার্ধ, পরিধি, গতি ইত্যাদি এবং চাকার বিহেভিয়ার হলো এটি ঘুরে। এখন যেহেতু আমরা সাইক্যাল এর চাকাকে কে আমরা প্রোগ্রামিং এর মাধ্যমে প্রকাশ করবো, সতুরাং এগুলোকে আমরা ভ্যারিয়েবল এ রাখবো। আর বিহেভিয়ার গুলোকে আমরা ফাংশন এর মাধ্যমে লিখি। আমরা এর আগে যাকে ফাংশন বলে এসেছি এখন থেকে আমরা ফাংশন কে ফাংশন বলবো না, আমরা এদেরকে মেথড বলবো।

সুতরাং আমরা জানলাম, অবজেক্ট এর দুইটা জিনিস থাকে, স্টেট (অর্থাৎ নিজের সম্পর্কে ধারণা) এবং মেথড (সে কি কি কাজ করতে পারে)।

ক্লাস**(Class)**

অবজেক্ট সম্পর্কে আমরা জানলাম। ক্লাস হলো সেই অবজেক্টটি তৈরি করার প্রক্রিয়ার একটি অংশ। মনে করি আমরা একটি কলম বানাতে চাই, শুরুতে আমরা কোন রকম চিন্তা ভাবনা না করে ফু দিয়ে একটা কিছু বানিয়ে ফেলতে পারি না। আমরা এর জন্যে পরিকল্পনা করি- কলমাটা দেখতে কেমন হবে, এটি লম্বা কতটুকু হবে, কলমটি কি কি কাজ করবে ইত্যাদি। এই পরিকল্পনা গুলো আমরা আমরা কোথাও লিখে রাখি। আমাদের এই লেখা ডকুমেন্টটি আসলে ক্লাস। সহজ একটি ব্যাপার।

অবজেক্ট ওরিয়েন্ট কনসেপ্ট তিনটি ধারণার উপর প্রতিষ্ঠিত।

এক, ইনহেরিট্যান্স- নাম থেকেই বুঝা যাচ্ছে যে এখানে বংশগতির একটা বিষয় চলে এসেছে। আসলেও তাই। ধরা যাক একটি একটা চাকা। নানা রকম চাকা হতে পারে, যেমন বাসের চাকা, সাইক্যাল এর চাকা, মটর সাইক্যাল এর চাকা ইত্যাদি। সব চাকার মধ্যেই কিন্তু কিছু কমন ব্যাপার আছে, এটির ব্যাসার্ধ আছে, পরিধি আছে এবং এটি ঘুরে। সুতরাং আমরা একটা চাকা নামের অবজেক্ট বানাতে পারি যা বাকি সব চাকা(বাস, সাইক্যাল এর) পূর্বপুরুষ। এতে আমাদের বেশি কিছু সুবিধা আছে, প্রধান সুবিধে হলো, কমন জিনিস গুলো নিয়ে আমাদের পূর্বপুরুষ তৈরি করার কাজ একবার করে ফেললেই হচ্ছে, উত্তরপুরুষ গুলোতে

আপনা আপনি সেই বৈশিষ্ট্যগুলো চলে আসবে।

দুই, এনক্যাপসুলেশান- মানে হলো জিনিসপত্র ক্যাপসুলের মধ্যে ভরে রাখা। আমরা অনেকেই ক্যাপসুল মেডিসিন খেয়েছি, এটির বাইরে একটা আবরণ দিয়ে সব কিছু ভেতরে আটকানো থাকে । ব্যাপারটি এমনি।

তিন, পলিমরফিজম – বহুরূপিতা। অর্থাৎ একি অঙ্গে নানা রূপ। একটা অবজেক্ট নানা সময় নানা রকম রূপ ধারণ করতে পারে।

তবে কেন এই অবজেক্ট ওরিয়েন্টেড কনসেপ্ট লাগবে সেটি নিয়ে প্রশ্ন হতে পারে। এবার তাহলে এই প্রশ্নের উত্তর ব্যাখ্যা করা যাক।

আমাদের পরিচিত প্রথাগত বা প্রসিডিউরাল প্রোগ্রামিং ল্যাংগুয়েজ যেমন- সি এর কিছু অসুবিধা রয়েছে। আমরা চাইলেই সহজে পুনরায় ব্যবহারযোগ্য কম্পোনেন্ট তৈরি করতে পারি না। সবচেয়ে বড় অসুবিধা হলো আমরা চাইলেই একটি প্রোগ্রাম থেকে একটি ফাংশান কপি করে অন্য একটি প্রোগ্রামে ব্যবহার করতে পারি না কারণ ফাংশন গুলো সাধারণত কিছু গ্লোবাল ভেরিয়েবল এবং অন্যান্য ফাংশন এর উপর নির্ভর করে। এই ল্যাংগুয়েজ গুলো হাই-লেভেল এবস্ট্রাকশান এর জন্যে মানানসই নয়। যেমন সি যে কম্পোনেন্ট গুলো ব্যবহার করে সেগুলো খুব লো-লেভেল-এর যা দিয়ে একটি বাস্তব জগতের সমস্যাকে খুব সহজে চিত্রায়ণ (portray) করা সম্ভব হয় না। কাস্টমার রিলেশনশিপ ম্যানেজমেন্ট বা সিআরএম অথবা ফুটবল খেলাকে সহজে সি দিয়ে চিত্রায়ণ করা কঠিন।

১৯৭০ সালের যুক্তরাষ্ট্রের প্রতিরক্ষা অদিদপ্তরের একটি টাস্কফোর্স তদন্ত করে বের করার চেষ্টা করে কেন আইটি(IT) বাজেট সবসময় নিয়ন্ত্রণ করা সম্ভব হয় না। সেগুলোর মধ্যে প্রধান গুলো এমন- ৮০% বাজেট শুধুমাত্র সফটওয়্যার এর জন্যে ব্যয় হয় আর বাকি ২০% ব্যয় হয় হার্ডওয়্যার এর জন্যে। এর মধ্যে ৮০% ব্যয় হয় শুধুমাত্র সফটওয়্যার মেইনটেইন করার জন্যে, বাকি ২০% তৈরি হয় সফটওয়্যার তৈরি করার জন্যে। হার্ডওয়্যার গুলো সহজেই রিইউজ বা পুনরায় ব্যবহার করা যায় এবং এতে এদের ইন্টিগ্রিটি নষ্ট হয় না, এবং একটি হার্ডওয়্যার একটি বিশেষ অংশ নষ্ট হয়ে গেলে তা সহজেই আলাদা করে ফেলা যায় এবং নতুন একটি দিয়ে রিপ্লেস করা যায়। কিন্তু সফটওয়্যার এর ক্ষেত্রে এমন সম্ভব হয় না, একটি প্রোগ্রাম এর সমস্যার জন্যে অন্য প্রোগ্রাম এর সমস্যা তৈরি হয় ইত্যাদি।

এই সমস্যা সমাধান করার জন্যে এই টাস্কফোর্স পরিশেষে প্রস্তাব করে যে সফটওয়্যার-ও হার্ডওয়্যার এর মতো হওয়া উচিৎ। পরবর্তীতে তারা তাদের সিস্টেম এর ৪৫০ টি প্রোগ্রামিং ল্যাংগুয়েজ রিপ্লেস করে এডা (Ada) নামে একটি অবজেক্ট ওরিয়েন্টেড প্রোগ্রামিং ল্যাংগুয়েজ ব্যবহার করে।

চলবে -------

ইনহেরিট্যান্স-

এবার আমরা অবজেক্ট ওরিয়েন্টেড কনসেপ্ট-এর আরও ভেতরে প্রবেশ করবো। শুরুতেই আমরা ইনহেরিটেন্স নিয়ো আলোচনা করি।

ইনহেরিটেন্স নিয়ে কথা বলতে হলে এর সাথে আরেকটি বিষয় চলে আসে সেটি হলো অবজেক্ট কম্পোজিশান। এটি মোটামুটিভাবে একটু কঠিন অন্যান্য টপিক থেকে। তাই এই টপিকটি পড়ার সময় একটু ধৈর্য্য নিয়ে পড়তে হবে।

তো শুরু করার যাক-

প্রথমেই আমরা কথা বলবো Is - A এবং Has – A নিয়ে।

যেহেতু আমরা জাভা প্রোগ্রামিং শুরু করেছি, তো আমরা যতই এর ভেতরে প্রবেশ করতে শুরু করবো, ততই বুঝতে শুরু করবো যে ক্লাস আসলে একটা স্ট্যান্ড এলোন কম্পোনেন্ট নয়, বরং এটি অন্যান্য ক্লাসের উপর নির্ভর করে। অর্থাৎ ক্লাস গুলো একটি রিলেশন মেইনটেইন করে চলে। এই রিলেশন গুলো সাধারণত দুই ধরণের হয়- Is - A এবং Has – A।

আমাদের বাস্তব জগৎ থেকে একটা এনালজি দেয়া যাক । যেমন একটি বিড়াল, কিংবা কার অথবা বাস। বিড়াল হচ্ছে একটি প্রাণি। কার এর থাকে চাকা এবং ইঞ্জিন। বাস এরও থাকে চাকা এব ইঞ্জিন। আবার কার এবং বাস দুটিই ভেহিকল বা যান।

এখানে যে উদাহরণ গুলো দেয়া হয়েছে এর সবগুলো মূলত Is - A অথবা Has – A রিলেশনশিপ মেইনটেইন করে । যেমন -

A cat is an Animal (বিড়াল একটি প্রাণি ।) A car has wheels (কার এর চাকা আছে ।)
A car has an engine (কার এর একটি ইঞ্জিন আছে ।)

তো ব্যপারটি একদম সহজ। ঠিক এই ব্যপারটিকে আমরা আমাদের অবজেক্ট ওরিয়েন্টেট কনসেপ্ট এর মাধ্যমে বলতে পারি। যখন কোন অবজেক্ট এর মাঝে Is – A এই সম্পর্কটি দেখবো তাকে বলবো ইনহেরিটেন্স। আবার যখন কোন অবজেক্ট এর মাঝে Has – A এই সম্পর্কটি দেখবো তখন সেই ব্যপারটিকে বলবো অবজেক্ট কম্পোজিশান।

ইনহেরিটেন্স মূলত একটি ট্রি-রিলেশানশিপ। অর্থাৎ এটি একটি অবজেক্ট থেকে ইনহেরিট করে আসে।

আর যখন আমরা অনেকগুলো অবজেক্ট নিয়ে আরেকটি অবজেক্ট তৈরি করবো তখন সেই নতুন অবজেক্ট হলো মেইড-আপ বা নতুন তৈরি করা অবজেক্ট এই ঘটনাটি হলো কম্পোজিশান।

এর সবই আসলে একটি কনসেপ্ট এবং আইডিয়া থেকে এসেছে, সেটি হলো কোড রিইউজ করা এবং সিম্পল করা। যেমন দুটি অবজেক্ট এর কোড এর কিছু অংশ যদি কমন থাকে তাহলে আমরা সেই অংশটিকে দুইটি ক্লাসের মধ্যে পুনরায় না লিখে বরং তাকে ব্যবহার করতে পারি।

ধরা যাক, আমরা দুটি অবজেক্ট তৈরি করতে চাই- Animal এবং Cat

আমরা জানি যে সব Animal খায়, ঘুমায়। সুতরাং আমরা এই ক্লাসে এই দুটি বৈশিষ্ট্য আমরা এই ক্লাসে লিখতে পারি। আবার যেহেতু আমরা জানি যে Cat হচ্ছে একটি Animal। সুতরাং আমরা যদি এমন ভাবে কোড লিখতে পারি, যাতে করে এই Cat ক্লাসের মধ্যে নতুন করে আর সেই দুটি বৈশিষ্ট্যের কোড আর লিখতে হচ্ছে না, বরং আমর এই Animal ক্লাসটিকে রিইউজ করলাম, তাহলে যে ঘটনাটি ঘটবে তাকেই মূলত ইনহেরিটেন্স বলা হয়।

এইভাবে আমরা আরও অন্যান্য Animal যেমন, Dog, Cow ইত্যাদি ক্লাস লিখতে পারি।

কম্পোজিশান তুলনামূলক ভাবে একটু সহজ।

যেমন আমরা একটি Car তৈরি করতে চাই। Car তৈরি করতে হলে আমাদের লাগবে Wheel এবং Engine. সুতরাং আমরা Wheel এবং Engine এই দুটি ক্লাসকে নিয়ে নতুন আরেকটি ক্লাস লিখবো।

এবার তাহলে একটি উদাহরণ দেখা যাক।

```
public class Bicycle {

    // the Bicycle class has three fields
    public int cadence;
    public int gear;
    public int speed;
```

```
    // the Bicycle class has one constructor
    public Bicycle(int startCadence, int startSpeed, int startGear) {
        gear = startGear;
        cadence = startCadence;
        speed = startSpeed;
    }

    // the Bicycle class has four methods
    public void setCadence(int newValue) {
        cadence = newValue;
    }

    public void setGear(int newValue) {
        gear = newValue;
    }

    public void applyBrake(int decrement) {
        speed -= decrement;
    }

    public void speedUp(int increment) {
        speed += increment;
    }
}
```

উপরের Bicycle ক্লাসটিতে তিনটি ফিল্ড এবং চারটি মেথড আছে। এবার এই Bicycle থেকে আমরা এর একটি সাব-ক্লাস লিখবো-

```
public class MountainBike extends Bicycle {

    // the MountainBike subclass adds one field
    public int seatHeight;

    // the MountainBike subclass has one constructor
    public MountainBike(int startHeight,
                        int startCadence,
                        int startSpeed,
                        int startGear) {
        super(startCadence, startSpeed, startGear);
        seatHeight = startHeight;
    }

    // the MountainBike subclass adds one method
    public void setHeight(int newValue) {
        seatHeight = newValue;
    }
}
```

এই MountainBike ক্লাসটি উপরে Bicycle এর সব ফিল্ড এবং মেথড গুলো ইনহেরিট করে এবং এতে নতুন করে শুধু একটি ফিল্ড এবং একটি মেথড লেখা হয়েছে। তাহলে আমাদের MountainBike ক্লাসটিতে Bicycle ক্লাসটির সব প্রোপার্টি এবং মেথড অটোম্যাটিক্যালি পেয়ে গেলো।

এখানে এ Bicycle হচ্ছে সুপার ক্লাস(Super Class) এবং MountainBike হচ্ছে সাব-ক্লাস(Sub Class) । অর্থাৎ যে ক্লাস থেকে ইনহেরিট করা হয় তাকে বলা হয় সুপার ক্লাস এবং যে ক্লাস সাব ক্লাস থেকে ইনহেরিট করে

মেথড অভাররাইডিং**(Method Overriding)**

যদিও সাব-ক্লাস সুপার-ক্লাসের সব গুলো প্রোপার্টি এবং মেথড ইনহেরিটর করে, তবে সাব-ক্লাসে সুপার ক্লাসের যে কোন প্রোপার্টি বা মেথড কে অভাররাইড করা যায়।

একটি উদাহরণ দেখা যাক-

```
  public class Circle {
      double radius;
      String color;

      public Circle(double radius, String color) {
          this.radius = radius;
          this.color = color;
      }

      public Circle() {
```

```
            radius = 1.0;
            color = "RED";
        }

        public double getArea() {
            return radius * radius * Math.PI;
        }
    }
```

এই ক্লাসটিতে `getArea()` মেথড একটি বৃত্তের ক্ষেত্রফল রিটার্ন করে।

এখন আমরা এই ক্লাসটিকে এক্সটেন্ড(`extends`) করে নতুন আরেকটি ক্লাস লিখবো-

```
public class Cylinder extends Circle {
    double height;

    public Cylinder() {
        this.height = 1.0;
    }

    public Cylinder(double radius, String color, double height) {
        super(radius, color);
        this.height = height;
    }

  @Override
    public double getArea() {
        return 2 * Math.PI * radius * height + 2 * super.getArea();
    }
}
```

এই ক্লাসটিতে `Circle` এর মেথডটি আমরা সাধারণ ভাবেই পেয়ে যাবো। `Cylinder` এর ক্ষেত্রফল নির্ধারণ করতে হলে `getArea()` কল করলেই হয়ে যাচ্ছে। কিন্তু আমরা জানি যে `Circle` এবং `Cylinder` এর ক্ষেত্রফল একভাবে নির্ধারণ করা যায় না। এক্ষেত্রে আমরা যদি `Circle` এর মেথডটি কে ব্যবহার করি তাহলে আমাদের ক্ষেত্রফলের মান ভুল আসবে। এই সমস্যা সমধাণ করার জন্যে আমরা আমাদের `Cylinder` ক্লাসটিতে `getArea()` মেথডটিকে পুনারায় লিখেছি।

এখানে লক্ষ্য রাখতে হবে যে, দুটি মেথড এর সিগনেচার, রিটার্ন-টাইপ এবং প্যারামিটার লিস্ট একই রকম হতে হবে।

এখন আমরা যদি `Cylinder` ক্লাস-এর `getArea()` মেথড কল করি, তাহলে অভারাইডেড মেথডটি কল হবে।

অ্যানোটেশান**(Annotation) @Override**

`@Override` এই অ্যানোটেশানটি জাভা 1.5 ভার্সনে প্রথম নিয়ে আসা হয়। কোন মেথডকে যদি আমরা অভাররাইড করি তাহলে সেই মেথড এর উপরে `@Override` দেয়া হয়। এটি কম্পাইলারকে ইনফর্ম করে যে, এই মেথটি সুপার ক্লাসের অভাররাইডেড মেথড।

তবে এটি অপশনাল হলেও অবশ্যই ভাল যদি ব্যবহার করা হয়।

`super` কিওয়ার্ড

আমরা যদি সাব ক্লাস থেকে সুপার ক্লাসের কোন মেথড বা ভেরিয়েবল একসেস করতে চাই তাহলে আমরা এই কিওয়ার্ডটি ব্যবহার করি। কিন্তু আমরা জানি যে সাব ক্লাসে অটোমেটিক্যালি সুপার ক্লাসের সব প্রোপার্টি চলে আসে তাহলে এর প্রয়োজনিয়তা নিয়ে প্রশ্ন হতে পারে।

আমরা আবার উপরের Cylinder ক্লাসটি আবার দেখি। এই ক্লাসটিতে আমরা নতুন আরেকটি মেথড লিখতে চাই।

```
    public double getVolume() {
        return getArea() * height;
    }
```

এই মেথডটি-তে `getArea() * height` এই স্ট্যাটমেন্টটি লক্ষ করি। এখানে `getArea()` এই মেথটি কল করা হয়েছে। আমাদের এই `Cylinder` ক্লাসটিতে `getArea()` মেথডটিকে আমরা `Circle` ক্লাস এর `getArea()` মেথড-কে অভারাইড করেছি। সুতরাং আমরা যখন এই `Cylinder` ক্লাস থেকে `getArea()` মেথডটি কল করবো তখন আসলে `Cylinder` ক্লাস এর মেথডটি কল হবে।

কিন্তু এক্ষেত্রে আমাদের একটি সমস্যা হচ্ছে যে - আমরা জানি সিলিন্ডারের আয়তন

```
V  = Pi * r^2 * h
   =  (Pi * r^2) * h
   =  Area of Circle * h
```

সুতরাং Cylinder ক্লাসের `getArea()` মেথডটি ব্যবহার করা যাচ্ছে না। কারণ সিলিন্ডারের ক্ষেত্রফল-

```
        A = (2 * Pi  * r * h)  +  (2 * Pi * r^2)
```

কিন্তু আমরা যদি `Circle` ক্লাস এর মেথডটি ব্যবহার করি তাহলে আমাদের সমস্যা সমাধান হয়ে যায়। এখন যদি আমরা সুপার ক্লাস(Circle) এর মেথডটি কল করে এই মেথডটি লিখতে চাই তাহলে -

```
      public double getVolume() {
            return super.getArea() * height;
      }
```

অর্থাৎ সাব ক্লাসে যদি মেথড অভাররাইড করা হয় এবং তারপরেও কোন কারণে যদি আমাদের সুপার ক্লাসের মেথড কে কল করার প্রয়োজন হয় তাহলে আমরা সুপার(`super`) কিওয়ার্ডটি ব্যবহার করি।

ইনহেরিটেন্স এর ক্ষেত্রে মনে রাখতে হবে –

জাভা মাল্টিপল ইনহেরিটেন্স সাপোর্ট করে না। এর মানে হচ্ছে আমার একটি ক্লাস শুধুমাত্র একটি ক্লাসকেই ইনহেরিট করতে পারে।

পলিমরফিজম (Polymorphism)

এবার আমরা কথা বলবো পলিমরফিজম নিয়ে। শব্দটির মধ্যেই একটি বিশেষ গাম্ভীর্য আছে যা কিনা একটি সাধারণ কথোপকথনকে অনেক গুরুত্বপূর্ণ করে তুলতে পারে। তবে এটি অবজেক্ট ওরিয়েন্টেড প্রোগ্রামিং এর একটি বহুল ব্যবহৃত কৌশল । এই শব্দটির সহজ মানে হচ্ছে যার একাধিক রূপ আছে অর্থাৎ বহুরূপিতা।

মনে করা যাক,

```
public class Liquid {
    public void swirl(boolean clockwise) {
        // Implement the default swirling behavior for liquids
        System.out.println("Swirling Liquid");
    }
}
```

এখন এর একটি অবজেক্ট তৈরি করতে চাইলে – আমাদের new অপারেটর ব্যবহার করে তা একটি ভেরিয়েবল এ রাখতে হবে।

```
    Liquid myFavoriteBeverage = new Liquid ();
```

এখানে `myFavoriteBeverage` হচ্ছে আমাদের ভেরিয়েবল যা `Liquid` অবজেক্ট এর রেফারেন্স। আমরা এখন পর্যন্ত যা যা শিখেছি সে অনুযায়ী এই স্টেন্টমেন্টটি যথার্থ। তবে আমরা এর আগের অধ্যায়ে `Is-A` সম্পর্কে জেনে এসেছি।

আমাদের জাভা প্রোগ্রামিং পলিমরফিজম সাপোর্ট করায় আমরা `myFavoriteBeverage` এই রেফারেন্সের যায়গায় `Is-A` সম্পর্কিত যে কোন টাইপ রাখতে পারি। যেমন –

```
Liquid myFavoriteBeverage = new Coffee();
Liquid myFavoriteBeverage = new Milk();
```

এখানে `Coffee` এবং `Milk` হচ্ছে `Liquid` এর সাব- ক্লাস বা টাইপ এবং `Liquid` এদের সুপার ক্লাস বা টাইপ।

পলিমরফিজম নিয়ে আরও একটু আশ্চর্য হতে চাইলে আমরা এখন একটি বিষয় জানবো যা দিয়ে আমরা কোন একটি অবজেক্ট এর কোন মেথড কল করবো তবে তা কোন ক্লাসের অবজেক্ট সেটি না জেনেই। আরেকটু পরিষ্কার করে বলি, আমরা যখন সুপার ক্লাসের এর রেফারেন্স ধরে কোন এর মেথড কল করবো তখন কিন্তু আমরা জানি না যে এটি আসলে কোন অবজেক্ট এর মেথড। যেমন-

```
Liquid myFavoriteBeverage = // ….
```

এখানে আমাদের myFavoriteBeverage এই রেফারেন্স এ `Liquid`, `Coffee`, `Milk` এর যেকোন একটির অবজেক্ট হতে পারে। উদাহরণ -

```
public class Coffee extends Liquid {
    @Override
    public void swirl(boolean clockwise) {
        System.out.println("Swirling Coffee");
    }
}

public class Milk extends Liquid{
    @Override
    public void swirl(boolean clockwise) {
        System.out.println("Swirling Milk");
    }
}

public class CoffeeCup {
    private Liquid innerLiquid;
```

```
        void addLiquid(Liquid liq) {
            innerLiquid = liq;
            // Swirl counterclockwise
            innerLiquid.swirl(false);
        }
    }
```

আমরা এখানে একটি `CoffeeCup` ক্লাস লিখেছি যার মাঝে `addLiquid()` নামে একটি মেথড আছে যা কিনা একটি `Liquid` টাইপ parameter নেয়, এবং সেই `Liquid` এর `swirl()` মেথড-কে কল করে।

কিন্তু আমরা আমাদের সত্যিকারের জগতে একটি কফি-কাপ এ শুধুমাত্র কফি-ই এড করতে পারি তা নয়, আমরা চাইলে যে কোন ধরণের লিকুইড এড করতে পারি, সেটি মিল্ক ও হতে পারে। তাহলে এই `addLiquid` মেথড তো শুধুমাত্র `Liquid` টাইপ parameter নেয়, তাহলে আমাদের সত্যিকারের জগতের সাথে এই প্রোগ্রামিং মডেল এর সাদৃশ্য থাকলো কোথায় ?

তবে মজার ব্যপার এখানেই, আমাদের এই `CoffeeCu`p ক্লাসটি পলিমরফিজমের ম্যজিক ব্যাবহার করে সত্যিকার অর্থেই আমাদের সত্যিকারের জগতের `CoffeeCup` এর মতোই কাজ করে।

```
    public class MainApp {
        public static void main(String[] args) {
            // First you need a coffee cup
            CoffeeCup myCup = new CoffeeCup();

            // Next you need various kinds of liquid
            Liquid genericLiquid = new Liquid();
            Coffee coffee = new Coffee();
            Milk milk = new Milk();

            // Now you can add the different liquids to the cup
            myCup.addLiquid(genericLiquid);
            myCup.addLiquid(coffee);
            myCup.addLiquid(milk);
        }
    }
```

উপরের কোড গুলোতে দেখা যাচ্ছে যে আমরা একটি `CoffeeCup` এর একটি অবজেক্ট তৈরি করে সেটি তে বিভিন্ন রকম `Liquid` এড করতে পারছি।

আরেকটু লক্ষ্য করি,

```
    void addLiquid(Liquid liq) {
            innerLiquid = liq;
            // Swirl counterclockwise
            innerLiquid.swirl(false);
      }
```

এই মেথডটিতে innerLiquid.swirl(false) যখন কল করি তখন কিন্তু আমরা জানি না যে এই innerLiquid আসলে কোন অবজেক্ট এর রেফারেন্স। এটি লিকুইড বা এর যে কোন সাব-টাইপ হতে পারে।

কিছু প্রয়োজনীয় তথ্য-

১. একটি সাব ক্লাস এর অবজেক্টকে আমরা এর সুপার ক্লাসের রেফারেন্স এ এসাইন করতে পারি। ২. সাব ক্লাসের অবজেক্টকে সুপার ক্লাসের রেফারেন্স-এ এসাইন করলে, মেথড কল করার সময় শুধু মাত্র সুপার ক্লাসের মেথড গুলোকেই কল করতে পারি। ৩. তবে সাব ক্লাস যদি সুপার ক্লাসের মেথড অভাররাইড করে, তাহলে যদিও আমরা সুপার ক্লাস এর রেফারেন্স ধরে মেথড কল করছি, কিন্তু রানটাইম-এ সাব ক্লাসের মেথডটি কল হবে। মনে রাখতে হবে এটি শুধুমাত্র মেথড অভাররাইড করা হলেই সত্য হবে।

আপ-কাস্টিং**(Upcasting)** এবং ডাউনকাস্টিং **(Downcasting)**

```
        Liquid liquid = new Coffee ();
```

এখানে সাব ক্লাসের অবজেক্টকে সুপার ক্লাসের রেফারেন্স এ এসাইন করা হয়ছে। একে বলা হয় আপ-কাস্টিং। এই কাস্টিং সবসময় সেইফ ধরা হয় কারণ আপকাস্টিং এর ক্ষেত্রে সাব ক্লাস সবসময়ই সুপার ক্লাসের সবকিছু ইনহেরিট করে এবং কম্পাইলার কম্পাইল করার সময়-ই এ কাস্টিং করা সম্ভব কিনা তা চেক

করে থাকে।

```
        Liquid liquid = new String();
```

উপরের স্টেটমেন্টটি কম্পাইলার কম্পাইল করবে না, কারণ `String` মোটেই `Liquid` ক্লাসের সাব ক্লাস নয়। এক্ষেত্রে কম্পাইলার incompatible types ইরর দেখাবে।

চলবে .....

ইনক্যাপসুলেশান (Encapsulation)

আমরা এতোক্ষনে জেনে ফেলেছি যে, একটি অবজেক্ট হলো কতগুলো ডাটা এবং মেথড এর সমষ্টি। অবজেক্ট ওরিয়েন্টেড প্রোগ্রামিং এর আরেকটি খুবই গুরুত্বপূর্ণ বিষয় আছে, যা হচ্ছে, একটি ক্লাসের মধ্যে ডাটা গুলোকে লুকিয়ে রাখা এবং শুধুমাত্র মেথডের মাধ্যমে সেগুলোকে একসেস করতে দেওয়া। এর নাম হচ্ছে এনক্যাপসুলেশান(Encapsulation) । এর মাধ্যমে আমরা সব ডাটা গুলোকে ক্লাসের মধ্যে সিল করে একটা কেপসুলের মধ্যে রেখে দিই এবং সেগুলো শুধুমাত্র যেসব মেথড গুলোকে ট্রাস্ট করা যায়, তাদের মাধ্যমে একসেস করতে দিই।

তবে এই এতো প্রোটেকশান এর কারণ কি হতে পারে তা যদি একটু জেনে নিই শুরুতে তাহলে আমার মনে খুব ভাল হয় –

যারা অনেক লেখালেখি করে এমনকি যারা কোড লিখে তারাও জানে যে, একটা লেখা ততই ভাল হয় সেটাকে যত বেশি রি-রাইট করা হয়। আপনি যদি একটা কোড লিখে ফেলে রাখেন এবং কিছুদিন পরে আবার সেটি খুলে দেখেন- দেখা যাবে যে আপনি আরও একটি ভাল উপায় বের পেয়ে যাবেন সেই কোডটি লেখার। এটি সব সময়ই হয়। এই বার বার কোড চেঞ্জ করে নতুন করে লেখাকে বলা হয় রিফেক্টরিং(refactoring)। আমরা একটি কোডকে বার বার লিখে সেটাকে আরও বেশি কিভাবে সহজবোধ্য কোড লেখার চেষ্টা করি যাতে সেই কোডটি আরও ভালভাবে মেইনটেইন করা যায়।

কিন্তু এখানে একটি চিন্তার বিষয় হচ্ছে। আমরা জানি যে অবজেক্ট ওরিয়েন্টেড প্রোগ্রামিং এর মাধ্যমে আমরা যে সফটওয়্যার সিস্টেম তৈরি করি তাতে নানা রকম অসংখ্য অবজেক্ট থাকে যেখানে একটি অবজেক্ট আরেকটির সাথে তথ্য আদান প্রদান করে, একটি আরেকটির উপর নির্ভর করে কাজ করে থাকে। ধরা যাক, A একটি অবজেক্ট যার উপর B নির্ভর করে। ধরা যাক B এখানে কনজ্যুমার অবজেক্ট। এখন আমারা যদি A কে কোন রকম পরিবর্তন করতে চাই, তাহলে B আগের মতোই থাকতে চাইবে। এখানে দুটো অবজেক্ট হয়তো দুইজন ভিন্ন প্রোগ্রামার লিখে থাকতে পারে। সুতরাং একে অন্যের পরিবর্তন নিয়ে যাতে সমস্যায় পরতে না হয়, সেই ব্যবস্থা করতে হবে। আমরা অনেক সময় নানা রকম লাইব্রেরি ব্যবহার করতে হয় বিভিন্ন প্রজেক্টে এবং এগুলোর উপর নির্ভর করে করে আমাদের প্রজেক্ট দাড়িয়ে যায়। এই লাইব্রেরি গুলোর মাঝেই মাঝেই ভার্সন পরিবর্তন হয়। কিন্তু মজার ব্যপার হলো এগুলো পরিবর্তন হলেও আমাদের কোড নতুন করে লিখতে হয় না। আবার অন্যদিকে লাইব্রেরি যারা তৈরি করে তাদেরও এই কোড পরিবর্তনের স্বাধীনতা থাকা চাই, কিন্তু সক্ষেত্রে যাতে আমাদের প্রজেক্ট এর কোন সমস্যা যাতে না হয় সেটাও মনে রাখতে হবে।

তো এই সমস্যা সমাধানের একটি উপায় আছে। সেটি হলো- লাইব্রেরি কোড-এর যে মেথড গুলো আছে সেগুলো মোটেও রিমুভ করা যাবে না। কারণ আমরা যখন একটি লাইব্রেরির একটি নির্দিষ্ট ক্লাসের মেথড নিয়ে কাজ করবো, আমরা চাইবো না কোন ভাবেই আমাদের কোড ভেঙ্গে যাক। লাইব্রেরির প্রোগ্রামার সেই ক্লাস নিয়ে যা কিছু করতে পারবে, কিন্তু আমরা যে সব মেথড ব্যবহার করেছি সেগুলোকে মুছে ফেলতে পারবে না। তারপর ফিল্ড বা প্রোপ্রার্টিজ এর ক্ষেত্রেও লাইব্রেরি যে লিখেছে সে কিভাবে জানবে যে কোন ফিল্ড বা প্রোপ্রার্টিজ গুলো আমরা আমাদের প্রজেক্ট এর ক্ষেত্রে একসেস করেছি? কোন ভাবেই জানার উপায় নেই। কারণ আমারা আমাদের কোড কিভাবে করেছি যা লাইব্রেরি যে লিখেছে তার জানার কথা নয়। কিন্তু যে প্রোগ্রামার লাইব্রেরি লিখেছে সে সবসময়ই চাইবে তার কোড এ নতুন কিছু এড করতে, আগের থেকে ভাল করা ইত্যাদি। এই সমস্যা সমাধানের জন্যে জাভা আমাদেরকে কতগুলো একেসেস স্পেসিফায়ার (access specifiers) দিয়ে থাকে, যার মাধ্যমে লাইব্রেরি প্রোগ্রামার ঠিক করতে পারে যে কোড এর কোন কোন অংশ গুলো আমরা যখন আমাদের প্রজেক্ট এ ব্যবহার করতে পারবো আর কোন কোন গুলো করতে পারবো না। এতে সুবিধা হচ্ছে, লাইব্রেরি প্রোগ্রামার সে সব অংশ গুলো আমাদেরকে ব্যবহার করতে দিচ্ছে, সেই অংশ গুলোতে ইচ্ছে মতো পরিবর্তন/পরিবর্ধন করতে পারবে কোন রকম চিন্তাভাবনা ছাড়া।

আমারা যখন একটা বড় সিস্টেমে কাজ করি আমাদের নানা রকম অবজেক্ট লিখতে হয়। একটি অবজেক্ট আরেকটি অবজেক্ট কে ব্যবহার করে। এই একেসেস প্রটেকশানের মাধ্যমে আমরা নির্ধারণ করে দিতে পারি যে একটি নির্দিষ্ট অবজেক্ট এর কোন অংশ গুলো অন্য অবজেক্ট ব্যভহার করতে পারবে ,আর কোন গুলো পারবে না। এতে উপরের সমস্যার সমাধান হয়ে যায়। এছাড়াও আরেকটি ব্যাপার হয়। আমরা যখন কোন একটি ক্লাস নিয়ে কাজ করতে যাবো, সেই অবজেক্ট-এ হাজার লাইন কোড থাকে পারে। পুরটা একেবারে দেখতে গেলে আমরা হয়তো কনফিউজড হয়ে যাবো কিংবা খুব কমপ্লেক্স কোড হলে বুঝতে অসুবিধা হতে পারে। কিন্তু সেই কোড যদি এমন ভাবে করা থাকে যেখানে অল্প অংশ আমাদের ব্যবহারের জন্যে অপেন করা থাকে, বাকি গুলো হাইড করা যাকে তাহলে আমরা যে অংশটুকু হাইড করা সেই অংশ নিয়ে চিন্তা করতে হবে না। এই কোড হাইড করার ঘটনাকে অবজেক্ট ওরিয়েন্টেড প্রোগ্রামিং এর ভাষায় এনক্যাপসুলেশান(Encapsulation) বলা হয়।

জাভাতে তিনটি একেসে কন্ট্রোল করার জন্যে তিনটি কি ওয়ার্ড আছে। সেগুলো হলে- `Public` , `protected` এবং `private` এখন আমরা বিভিন্ন রকম একসেস কন্ট্রোল দেখবো-

Default Access

এর মানে হচ্ছে আমরা যদি কোন কি-ওয়ার্ড ব্যবহার না করি তাহলে সেটি Default Access আর মাঝে পরে। কোন ক্লাস এর ভেরিয়েবল বা মেথড এর আগে যদি কোন একসেস মডিফায়ার না থাকে তাহলে সেই ক্লাসটি যে প্যাকেজের মধ্যে আছে সেই প্যাকেজ এর সব ক্লাস থেকে একসেস করা যাবে।

```
package bd.com.howtocode.java;

import java.util.Random;

public class HelloWorld {
    String version = "2.56";

    int getRandomInt() {
```

```
        return new Random().nextInt();
    }
}
```

এই ক্লাসের ভেরিয়েবল version এবং getRandomInt() মেথড কে bd.com.howtocode.java এই প্যাকেজ এর সকল ক্লাস একসেস করতে পারবে।

Private Access Modifier - `private` :

কোন ক্লাসের মেথড, ভেরিয়েবল, কনস্ট্রাকটর এর আগে যদি private কিওয়ার্ড থাকে তাহলে সেগুলোকে সেই ক্লাস ছাড়া অন্য কোন ক্লাস একসেস করতে পারবে না।

উদাহরণ-

```
package bd.com.howtocode.java;

public class User {
    private String name;
    private String emailAddress;

    public String getName() {
        return name;
    }

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

    public String getEmailAddress() {
        return emailAddress;
    }

    public void setEmailAddress(String emailAddress) {
        this.emailAddress = emailAddress;
    }
}
```

এই ক্লাসের এর ভেরিয়েবল `name` এবং `emailAddress` কে কোন ভাবেই অন্য কোন ক্লাস থেকে একসেস করা যাবে না। কিন্তু আমরা যদি এদের কে একসেস করতে চাই তাহলে একসেসর মেথড ব্যাবহার করতে পারি।

Public Access Modifier - `public` :

কোন ক্লাসের মেথড, ভেরিয়েবল, কনস্ট্রাকটর এর আগে যদি public কিওয়ার্ড থাকে তাহলে সেগুলোকে যে কোন ক্লাস থেকে একসেস করা যায়।

```
public class Milk{
    public void swirl(boolean clockwise) {
        System.out.println("Swirling Milk");
    }
}
```

Protected Access Modifier - `protected` :

কোন ক্লাসের মেথড, ভেরিয়েবল, কনস্ট্রাকটর এর আগে যদি protected কিওয়ার্ড থাকে তাহলে সেগুলোকে অন্য প্যাকেজ থেকে সেই ক্লাসের সাব ক্লাস একসেস করতে পারবে আর নিজের প্যাকেজ এর সবাই একসেস করতে পারবে।

```
class AudioPlayer {
   protected boolean openSpeaker(Speaker sp) {
      // implementation details
   }
}
```

একসেস লেভেল একটি টেবলি -

Modifier	Class	Package	Subclass	World
public	Y	Y	Y	Y
protected	Y	Y	Y	N
no modifier	Y	Y	N	N
private	Y	N	N	N

পাঠ ৬: জাভা এক্সেপশান হ্যান্ডেলিং

- এক্সেপশান বেসিকস
- ট্রাই ক্যাচ- ফাইনালি
- চেকড-একসেপশান
- আনচেকড-একসেপশান
- থ্রুয়িং একসেপশান
- NullPointerException
- ArrayIndexOutOfBoundsException
- কিভাবে নিজস্ব এক্সেপশান লিখবো
- সারসংক্ষেপ

আমরা একটি প্রোগ্রাম লিখি যার একটি নরমাল ফ্লো থাকে, তবে কোন কারণে যদি এই ফ্লো ব্যাহত হয় তাহলে জাভা রানটাইম একটি ইভেন্ট ফায়ার করে, একে এক্সেপশান বলা হয়।

সহজ কথায় এক্সেপশন হচ্ছে এক ধরণের ইরর যা কিনা প্রোগ্রাম চলাকালীন সময়ে দেখা দিতে পারে ।

একটি উদাহরণ দেখা যাক-

```
public class Main {

    public static void main(String[] args) {
        int a = 1;
        int b = 0;

        int result = divide(a, b);   // 1
        System.out.println("Result: " + result);  // 2
    }

    public static int divide(int a, int b) {
        return a / b;
    }
}
```

১. এখানে `divide()` মেথডটিতে a এবং b আর্গুমেন্ট পাস করা হলে মেথডটি প্রথম আর্গুমেন্টকে দ্বিতীয় আর্গুমেন্ট দিয়ে ভাগ করে ফলাফল `result` ভ্যারিয়েবল-টিতে এসাইন করবে।

২. এখানে `result` এর মান প্রিন্ট করা হবে।

আমরা যদি এই প্রোগ্রামটি রান করি তাহলে console এ নিচের আউটপুট-টি পাবো-

```
Exception in thread "main" java.lang.ArithmeticException: / by zero
    at com.bazlur.exception.Main.divide(Main.java:18)
    at com.bazlur.exception.Main.main(Main.java:13)
    at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
    at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:62)
    at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)
    at java.lang.reflect.Method.invoke(Method.java:483)
    at com.intellij.rt.execution.application.AppMain.main(AppMain.java:134)
```

এই আউটপুট থেকে আমরা বুঝতে পারি যে, আমাদের প্রোগ্রামটি-তে একটি সমস্যা হয়েছে এবং প্রোগ্রামটি এখানেই থেমে গেছে, `System.out.println("Result: " + result);` এই লাইনটি এক্সিকিউট হয় নি।

এবার আমরা নিচের প্রোগ্রামটি রান করি-

```
public class Main {
```

```
    public static void main(String[] args) {
        int a = 1;
        int b = 0;

        int result = 0;
        try {
            result = divide(a, b);
        } catch (ArithmeticException e) {
            System.out.println("You can't divide " + a + " by " + b);
        }

        System.out.println("Result: " + result);
    }


    public static int divide(int a, int b) {
        return a / b;
    }
}
```

এবার **console** এ নিচের আউটপুটটি দেখবো -

> You can't divide 1 by 0
>
> Result: 0

এবার লক্ষ্য করুন। প্রোগ্রামটি কিন্তু থেমে যাই নি, বরং শুণ্য দিয়ে যে কোন সংখ্যাকে ভাগ করা যাবে না, তার জন্য একটি মেসেস প্রিন্ট করেছে এবং শেষ পর্যন্ত প্রত্যেকটি লাইন এক্সিকিউট হয়েছে।

এই প্রোগ্রামটিতে আমরা নতুন কিওয়ার্ড ব্যবহার করেছি, সেগুলো হলো- try, catch এবং এগুলো দিয়ে আমাদের যে কোড ব্লকটিতে ইরর হওয়ার সম্ভবনা ছিল, সেই অংগটুকুকে wrap করেছি। এতে করে এই কোড ব্লক-এ যদি কোন ধরণের ইরর হয় তাহলে প্রোগ্রামটি catch ব্লক-এ চলে যায়, এবং এই ব্লক এর ইন্সট্রাকশন গুলো এক্সিকিউট করে এরপর নিচের কোড ব্লক এ চলে যায়।

আর এই প্রক্রিয়াকে আমরা এক্সেপশন হ্যান্ডেলিং বলি, অর্থাৎ প্রোগ্রাম এর কোন অংশে যদি কোন ধরণের এক্সেপশন বা ইরর হয় তাহলে আমাদের প্রোগ্রামটি যাতে বন্ধ না হয় যায় বরং সেইসব অবস্থায় ইউজারকে যাতে করে অর্থপূর্ণ মেসেস দেওয়াকে এক্সেপশন হ্যান্ডেলিং বলে।

The try Block

যদি কোন কোড ব্লক -এ যদি ইরর হওয়ার সম্ভবনা থাকে তাহলে আমরা সেই কেড ব্লক-কে `try` ব্লক দিয়ে ইনক্লাজে করতে হয়। উদাহরণ-

```
try {
    code
}
catch and finally blocks . . .
```

এই `try` ব্লক এর মাঝে এক বা একাধিক লাইন কোড থাকতে পারে। catch এবং finally ব্লক পরের সেকশনে দেখানো হবে।

একটি উদাহরণ দেখা যাক-

```
private List<Integer> list;
    private static final int SIZE = 10;

    public void writeList() {
        PrintWriter out = null;
        try {
            System.out.println("Entered try statement");
            out = new PrintWriter(new FileWriter("file.txt"));
            for (int i = 0; i < SIZE; i++) {
                out.write(i);
            }

        } catch (IOException e) {
        }
    }
```

উপরের প্রোগ্রামটিতে একটি মেথড আছে - writeList() যা কিনা একটি ফাইল এ একটি লিস্ট থেকে ভ্যালু পড়ে তা রাইট করে। এই মেথড-টি তে একাধিক এক্সেপশান বা ইরর হতে পারে। যেমন -
`out = new PrintWriter(new FileWriter("file.txt"));` এই লাইনটিতে আমরা একটি ফাইল অপেন করার চেষ্টা করেছি। কিন্তু এই ফাইলটি সিস্টেমে নাও থাকতে পারে, কিংবা থাকলেও সেটি অপেন করা যাচ্ছে না ইত্যাদি। সেক্ষেত্রে আমাদরে সিস্টেম `IOException` থ্রু করবে এবং প্রোগ্রামটি বন্ধ হয়ে যাবে। এছাড়াও আমরা একটি ফর লুপ ব্যবহার করেছি, এক্ষেত্রে ফাইল এ রাইট করার সময়ও ইরর বা এক্সেপশন হতে পারে। তাই এইসব ইরর বা এক্সেপশন কে হ্যান্ডেল করার জন্যে আমরা কোড ব্লকটিকে `try` ব্লক এর ভেতরে রেখেছি। এখন প্রোগ্রামটি চলার সময় যদি কোন ইরর বা একসেপশন হয় তাহলে প্রোগ্রাম এক্সিকিশান সেখান থেকেই `catch` ব্লক এ চলে যাবে।

The catch Blocks

try ব্লক এর সাথেই catch ব্লক লিখতে হয়। তবে আমরা একটি try ব্লকের সাথে একাধিক catch ব্লক লিখতে পারি। উদাহরণ-

```
try {

} catch (ExceptionType name) {
//  catch blog # 1
} catch (ExceptionType name) {
//  catch blog # 1
}
```

`catch` কিওয়ার্ড এর সাথে প্যারেন্থেসিস এর মাঝে আমরা আর্গুমেন্ট দিতে হয় যা কি টাইপ এক্সেপশন হ্যান্ডেল করা হচ্ছে তা নির্দেশ করে। এখানে ExceptionType একটি প্লেস হোল্ডার । এখানে যে কোন ক্লাস যা কিনা Throwable ক্লাস কে ইনহেরিট করে তা বসতে পারে।

try ব্লক এর কোন কোড-এ যদি কোন এরর বা এক্সিসেপশন হয় তাহলে প্রেগ্রামের এক্সিকিউশান পয়েন্ট `catch` ব্লকে চলে আসে এবং শুধুমাত্র তখনি `catch` ব্লক এর কোড এক্সিকিউট হয়।

যদি একাধিক `catch` ব্লক থাকে তাহলে এক্সেপশন এর টাইপ অনুযায়ী `catch` ব্লক সিলেকটেড হয়।

```
try {

} catch (IndexOutOfBoundsException e) {
    System.err.println("IndexOutOfBoundsException: " + e.getMessage());
} catch (IOException e) {
    System.err.println("Caught IOException: " + e.getMessage());
}
```

এখানে `try` ব্লকে যদি `IndexOutOfBoundsException` হয় তাহলে প্রথম `catch` ব্লকটি এক্সিকিউট হবে । আর যদি IOException হয় তাহলে পরের `catch` ব্লকটি এক্সিকিউট হবে।

জাভা ৭ এবং পরবর্তি ভার্সন গুলোর জন্যে একটি নতুন ফিচার আছে যাতে করে একটি `catch` ব্লক দিয়ে অনেকগুলো এক্সেপশন হ্যান্ডেল করা যায়। উহারহণ -

```
catch (IOException|SQLException ex) {
    logger.log(ex);
}
```

এখানে catch ব্লজ-এ একাধিক এক্সেপশান একটি ভার্টিকেল বার (|) দিয়ে আলাদা করা হয়।

The finally Block

উপরের উদাহরণ গুলো থেকে দেখলাম যে , `try` ব্লক এর কোড -এ এক্সেসেপশন হলে শুধুমাত্র `catch` ব্লকের কোড গুলো এক্সিকিউট হয়। তবে আমাদের এমন কোন সিচুয়েশন থাকতে পারে যখন আমরা চাই ইরর হোক বা না হোক, একটি কোড ব্লক আমরা সবসমই এক্সিকিউট করতে চাই , তাহলে আমরা finally ব্যবহার করি।

```
public void openFile() {
        FileReader reader = null;
        try {
```

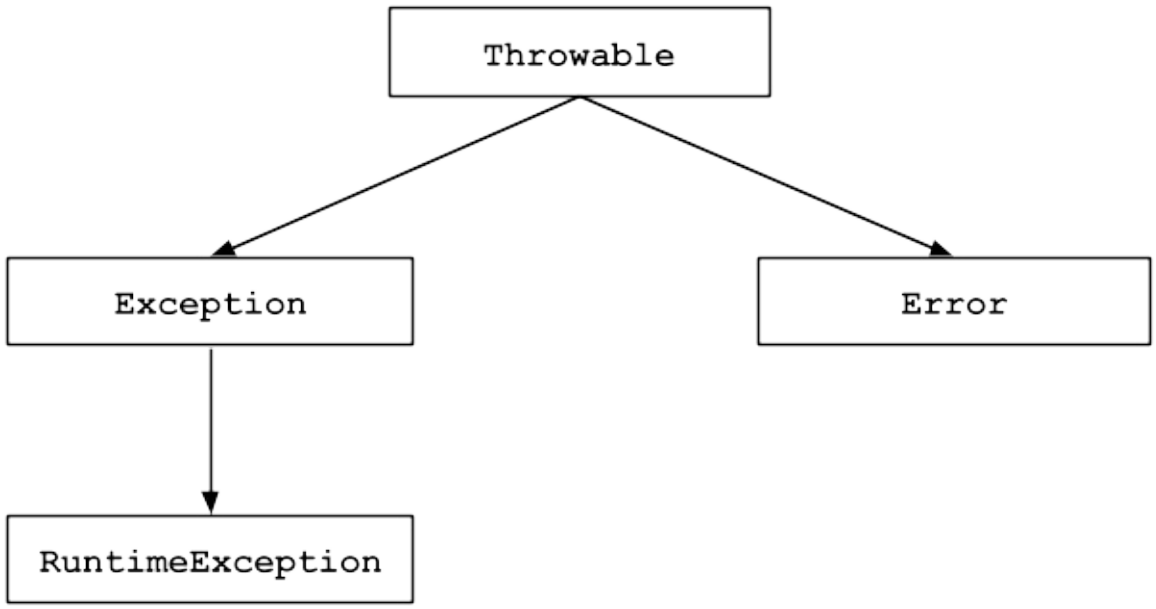
```
            reader = new FileReader("someFile");
            int i = 0;
            while (i != -1) {
                i = reader.read();
                System.out.println((char) i);
            }
        } catch (IOException e) {
            //do something clever with the exception
        } finally {
            if (reader != null) {
                try {
                    reader.close();
                } catch (IOException e) {
                    //do something clever with the exception
                }
            }
            System.out.println("--- File End ---");
        }
    }
```

উপরের প্রোগ্রামটি তে আমরা একটি ফাইল অপেন করছি এবং কিছু কাজ করেছি। এজন্যে একটি FileReader ক্লাসের অবজেক্ট তৈরি করেছি। আমরা চাই এই FileReader অবজেক্টি কাজ শেষ হয়ে গেলে ক্লোজ করতে। এক্ষেত্রে আমরা finally ব্লক এ আমাদের একই ক্লোজিং এর কোডটি লিখেছি। এতে করে এই সুবিধা হচ্ছে যে, আমাদের এই ট্রাই ব্লক-এর কোড কাজ করুক আর না করুক, শেষে আমাদের FileReader এর অবজেক্টটি ক্লোজ হয়ে যাচ্ছে।

অর্থাৎ আমরা শুধুমাত্রে তখনি ফাইনালী ব্লক ব্যবহার করি যখন আমরা নো ম্যাটার হুয়াট, একটি কোড ব্লক সবসময়ই এক্সিকউট করতে চাই।

Checked or Unchecked Exceptions

জাভাতে সব এক্সেপশান গুলো Throwable ক্লাসকে ইনহেরিট করে তৈরি। অর্থাৎ এক্সেপশান হাইআরকি এর একদপ উপরে এই Throwable ক্লাস এর অবস্থান। এর ঠিক নিচেই দুটি সাব ক্লাস হলে - Exception এবং অন্যটি হলো RuntimeException । এবং এই দুটি ক্লাস দুটি আলাদা শ্রেণীবিভাগের সূচনা করেছে। তবে এই শ্রেণীবিভাগের আরেকটি শাখা আছে, সেটি হলো - Error তবে এগুলো প্রোগ্রাম চলাকালিন সময়ে সাধারণত ধরা হয় না। এগুলো মূলত জাভা রানটাইম সিস্টেম নিজে থেকে হ্যান্ডেল করে এবং এটি আমাদের এই বইয়ের আলোচনার বাইরে।




```
public class ExceptionDemo5 {

    public void fetchData(String url) {
        try {
            String data = fetchDataFromUrl(url);
        } catch (CheckedException e) {
            e.printStackTrace();
        }
    }

    public String fetchDataFromUrl(String url) throws CheckedException {
```

```
        if (url == null) {
            throw new CheckedException("Url Not found");
        }

        String data = null;
        //read lots of data over HTTP and return
        //it as a String instance.

        return data;
    }
}
```

```
public class ExceptionDemo6 {
    public void fetchData(String url) {
        String data = fetchDataFromUrl(url);
    }

    public String fetchDataFromUrl(String url) {
        if (url == null) {
            throw new UncheckedException("Url Not found");
        }

        String data = null;
        //read lots of data over HTTP and return
        //it as a String instance.

        return data;
    }
}
```

```
public class CheckedException extends Exception {
    public CheckedException(String message) {
        super(message);
    }
}
```

```
public class UncheckedException extends RuntimeException {
    public UncheckedException(String message) {
        super(message);
    }
}
```

পাঠ ৭: স্ট্রিং অপারেশান

- স্ট্রিং তেরি করা
- স্ট্রিং লেন্থ এবং স্ট্রিং অপারেশান
- ক্যারেকটার এক্সট্রাকশন
- স্ট্রিং কমপেরিঝন
- স্ট্রিং সার্চিং এবং মডিফাইং
- ডাটা কনভারশন
- স্ট্রিং বাফার
- স্ট্রিং বিউল্ডার
- সারসংক্ষেপ

জাভাতে স্ট্রিং ব্যাপকভাবে ব্যবহৃত একটি অবজেক্ট। স্ট্রিং হচ্ছে কতগুলো ক্যারেক্টার-এর সিকুয়েন্স বা অনুক্রম।
স্ট্রিং তৈরি করা খুব সহজ। যেমন –

```
String greeting = "Hello world!";
```

এখানে "Hello world!" হচ্ছে স্ট্রিং লিটারেল যা অকনেগুলো ক্যারেক্টার উদ্ধৃতি চিহ্নের (“”) মাঝে লিখতে হয়।

জাভা কোডের মধ্যে কোন স্ট্রিং লিটারেল থাকলে কম্পাইলার সেটিকে String অবজেক্ট –এ পরিণত করে যার ভ্যালু হয় উদ্ধৃতি চিহ্নের (“”) মাঝের ক্যারেক্টার গুলো ।

তবে অন্যান্য অবজেক্ট এর মতো String ও new কিওয়ার্ড এবং কন্সট্রাকটর ব্যবহার করে তৈরি করতে পারি। String ক্লাসের ১৩ টি কনস্ট্রাকটর আছে। সুতরাং আমরা আরও ১৩ টি উপায়ে স্ট্রিং তৈরি করতে পারি।

উদাহরণ –

```
String str = new String("Hello world!");
char[] helloArray = { 'h', 'e', 'l', 'l', 'o', '.' };
String helloString = new String(helloArray);
```

String Length

String ক্লাসের মধ্যে length() মেথড থাকে যা একটি স্ট্রিং এর মধ্যে কতগুলো ক্যারেক্টার থাকে তার সংখ্যা রিটার্ণ করে। String loremIpsum ="Lorem ipsum dolor sit amet.";

```
int len = loremIpsum.length();
```

স্ট্রিং Concatenating

আমরা বেশ কয়েকটি উপায়ে স্ট্রিং কনকেট করতে পারি -

```
string1.concat(string2); //  concat()মেথড ব্যবহার করে
"My name is ".concat("Rumplestiltskin"); // কনটারেন ব্যবহার করে
"Hello," + " world" + "!" //  + অপারেটর ব্যবহার করে
```

স্ট্রিং এর ভেতর বেশ কিছু মেথড আছে যেগুলো ব্যবহার করে আমরা স্ট্রিং মেনুপুলেট করতে পারি।

charAt() – এই মেথড ব্যবহার করে আমরা কোন ইনডেক্স এর ক্যারেক্টার আলাদা করতে পারি। উদাহরণ-

```
    String hello = "Hello";
    char getCharOfIndex2 = hello.charAt(2);
```

substring() – এই মেথড ব্যবহার করে আমরা একটি স্ট্রিং থেকে এর সাব-স্ট্রিং বা কোন নির্দিষ্ট অংশ আলাদা করতে পারি। উদাহরণ-

```
    String str1 = "Hello world!";
    String hello =str1.substring(0,5);
```

toLowerCase() – লোওয়ারকেস লেটারে কনভার্ট করার জন্যে এই মেথড ব্যবহার করি। toUpperCase() আপারকেস লেটারে কনভার্ট করার জন্যে এই মেথড ব্যবহার করি।

উদাহরণ –

```
      String hello = "Hello";
      hello.toUpperCase(); // HELLO
      hello.toLowerCase(); // hello
```

নিচে আরও কিছু উদহরণ দেখানো হল-

```
      String str2 = "Hello world!";
      int indexOfHaitch = str2.indexOf("H");
```

বিশেষভাবে লক্ষণীয়

জাভাতে স্ট্রিং ক্লাস **immutable**, এর মানে হচ্ছে, একবার কোন স্ট্রিং অবজেক্ট তৈরি করলে তাকে আর পরিবর্তন করা যাবে না। আমরা অনেক ক্লাস লিখি, তারপর এর মাঝে বিভিন্ন ভ্যারিয়েবল রাখি, অবজেক্ট তৈরি করার পর সেই অবজেক্টর এর ভেতরের ভ্যারিয়েবল গুলো বিভিন্ন সময় পরিবতর্তন করতে পারি। কিন্তু স্ট্রিং এর ক্ষেত্রে এটি সম্ভব নয়। অর্থাৎ আমরা যদি কোন একটি ভ্যালু দিয়ে একবার একটি স্ট্রিং অবজেক্ট তৈরি করি তাহলে সেটি আর পরিবর্তন করা যাবে না।

কিন্তু আমরা অনেকসময়ই স্ট্রিং কনকেট করি, সেক্ষেত্রে যা হয়, মনে করি-

```
    String str = "Hello ";
    str = str + "world";
`
```

এখানে যদিও মনে হচ্ছে আমরা স্ট্রিং এর ভ্যালু পরিবর্তন করে ফেলেছি। কিন্তু আসলে যা হচ্ছে তা হলো, আমরা প্রথমে একটি অবজেক্ট তৈরি করেছি, তারপর সেই অবেজক্টএর ভ্যালু এবং নতুন একটি ভ্যালু নিয়ে নতুন একটি অবজেক্ট তৈরি করেছি, এবং যা str এখন নতন সেই অবজেক্টকে রেফার করছে। আগের অবজেক্টটিকে গার্বেজ কালেক্টর নিয়ে চলে যাবে।

এখন প্রশ্ন হচ্ছে, কেন এই **immutability** দরকার হয়।

স্ট্রিং পুল (**Stirng Pool**) সম্পর্কে হয়তো অনেকেই জানি। এটি একটি জাভা হিপ এর একটি স্পেশাল এরিয়া । আমাদের যদি নতুন একটি স্ট্রিং তৈরি করতে হয়, সেই স্ট্রিং যদি আগে থেকেই স্ট্রিং পুল এ থেকে থাকে, তাহলে নতুন করে আর তৈরি না করে আগের অবজেক্টটির রেফারেন্স দেওয়া হয়। এতে করে আমাদের মেমরি ফুটপ্রিন্ট অনেক কমে যাচ্ছে।

```
String string1 = "abcd";
String string2 = "abcd";
`
```

আমরা যদি এই দুটি লাইন লিখি, তাহলে আসলে জাভা হিপ এ একটি স্ট্রিং অবজেক্ট-ই থাকবে, দুটি তৈরি হবে না। যদি স্ট্রিং immutable না হয়, তাহলে একটি স্ট্রিং যদি পরিবর্তন করি, তাহলে আসলে অন্যান্য রেফারেন্স গুলোও পরিবর্তন হয়ে যাবে।

এছাড়াও, আমরা জানি যে স্ট্রিং এর **hashcode** খুব বেশি ব্যবহার করা হয়। যেমন **HashMap**। স্ট্রিং **immutable** হওয়ায় এটা গ্যারান্টিড যে, সবসময় **hashcode** এক-ই হবে, সুতরাং আমরা প্রতিবার **hashcode** ক্যালকুলেট না করে নির্দ্বিধায় ক্যাশিং করতে পারি।

আমরা স্ট্রিং প্যারামিটার হিসেব অনেক বেশি ব্যবহার করে থাকি, যেমন, নেটওয়ার্ক কানেকশান, ফাইল অপেনিং ইত্যাদির ক্ষেত্রে। সুতরাং এটি **immutable** না হলে পরিবর্তন করে ফেলা সম্ভব যা কিনা একটি সিরিয়াস রকম সিকিউরিটি থ্রেড হতে পারে। কিন্তু যেহেতু স্ট্রিং **immutable**, সুতরাং সেই সম্ভবনা নেই।

তাছাড়া স্ট্রিং **immutable** হওয়ায় এটি ন্যাচারালি থ্রেড সেইফ, এবং স্বাধীনভাবে যে কেন থ্রেড একসেস করে পারে আমাদেরকে কষ্ট করে এর থ্রেড সেইফটি নিয়ে চিন্তা করতে হয় না।

চলবে .....

পাঠ ৮: জেনেরিকস

জেনেরিকস ইন জাভা (Generics in Java)

আমরা জাভা-এর টাইপ সিস্টেম সর্ম্পকে জানি। আমরা জানি জাভাতে কোন প্রোগ্রাম লিখতে হলে আমাদের কে টাইপ বলে দিতে হয় । যেমন আমরা যদি একটি মেথড লিখি তাহলে মেথডটি কি টাইপ প্যারামিটার এক্সেপ্ট করবে তা বলে দিতে হয়।

তবে জাভাতে একটি চমৎকার ফিচার আছে যাতে করে আমরা অনেক সময় টাইপ না বলে দিয়েই কোড লিখতে পারি। আমরা জেনেরিকস শুরু করার আগে একটি গুরুত্বপূর্ণ তথ্য জেনে নিই- জাভা প্রোগ্রামিং ল্যাংগুয়েজ এ সব ক্লাস **java.lang.Object** ক্লাসটিকে ইনহেরিট করে। আমরা এটি নিয়ে অন্য কোন চ্যাপ্টারে আলোচনা করবো, তবে এখন আমাদের শুধু এই তথ্যটুকু মনে রাখলেই চলবে।

সহজ কথায় যদি বলি, তাহলে জেনেরিকস দিয়ে আমরা যখন অবজেক্ট তৈরি করবো তখন টাইপ প্যারিমিটারাইজ করতে পারি। অর্থাৎ আমরা যখন `new` অপারেটর দিয়ে অবজেক্ট তৈরি করবো তখন আসলে সিদ্ধান্ত নেবো এটির টাইপ কি হবে। এর আগে আমরা এমন ভাবে একটা ক্লাস বা মেথড লিখে ফেলতে পারি যাতে করে এটি যে কোন টাইপ এর জন্যে কাজ করে।

বরং একটা উদাহরণ দেখা যাক-

```
//একটি মিশ্রন ক্লাস , এখানে  T হচ্ছে টাইপ প্যারামিটার যা অবজেক্ট তৈরি করার সময় রিয়েল টাইপ দিয়ে রিপ্লেস হবে
public class Generic<T> {
    T obj;
 // একটা টাইপ ভ্যারিয়বল ডিক্লেয়ার করা হলো


   // কনস্ট্রাক্টর - যে একটি রিয়েল অবজেক্ট আর্গুমেন্ট হিসেবে নেয়
    public Generic(T obj) {
        this.obj = obj;
    }

//  অবজেক্টটি একসেস করার জন্যে একটি মেথড
    public T getObj() {
        return obj;
    }

   // রানটাইমে অবজেক্ট-এর টাইপ আসলে কি , তা প্রিন্ট করে দেখি
    public void showType() {
        System.out.println("Type of T is: " + obj.getClass().getName());
    }


    public static void main(String[] args) {

    // একটি ইন্টিজার এর রেফারেন্স
    Generic<Integer> iObj;

    // অবজেক্ট তৈরি করি এবং iObj রেফারেন্স এ এসাইন করি এবং কনস্ট্রাক্টর আর্গুমেন্ট হিসেবে 88 পাস করি
        iObj = new Generic<Integer>(88);

    // রানটাইম-এ তাহলে জেনেরিক ক্লাসটিতে T obj একটি ইন্টিজার হয়ে যাওয়ার কথা, প্রিন্ট করে দেখা যাক
    iObj.showType();


    int v = iObj.getObj();
 // ইন্টিজার ভ্যালুটি এর ভ্যালু একসেস ককরে v তে রাখা হল

    System.out.println("value: " + v);
 // প্রিন্ট করি, যেথা যাক, আমরা এর ভ্যালু ঠিক ঠাক মতো পাওয়া যায় কিনা


//এভাবে আমরা একটি স্ট্রিং টাইপ দিয়েও পরীক্ষা করতে পারি।
        Generic<String> strOb = new Generic<String>("This is a Generics Test");
        strOb.showType();
        String str = strOb.getObj();
        System.out.println("value: " + str);
    }
}
```

এই প্রোগ্রামটি যদি রান করা হয়, তাহলে নিচের আউটপুট গুলো দেখা যাবে -

> Type of T is: java.lang.Integer value: 88 Type of T is: java.lang.String value: This is a Generics Test

আউটপুট গুলো থেকে বুঝা যাচ্ছে যে , আমাদের প্রোগ্রামটি সঠিক ভাবে কাজ করছে এবং একটি জেনেরিক ক্লাসে একটি ইন্টিজার এবং একটি স্ট্রিং প্যারামিটারাইজড করতে পেরেছি।

এভাবে আমরা আরও অন্যান্য টাইপ-ও প্যারামিটারাইজড করে পারি।

এবার আরও ভালভাবে এই প্রোগ্রামটি খেয়াল করা যাক-

```
    public class Generic<T> {
    }
`
```

এখানে `T` হচ্ছে টাইপ প্যারামিটার। এটি মূলত একটি প্লেস হোল্ডার।

লক্ষ্য করুন – এর `T` কিন্তু `<>` এর মধ্যে থাকে।

আমরা সাধারণত যেভাবে ভ্যারিয়েবল ডিক্লেয়ার করি, সেভাবেই আমরা জেনেরিক্স-এ ভ্যারিয়েবল ডিক্লেয়ার করতে পারি। এর জন্যে আলাদা কোন নিয়ম নেই।

```
     T obj;
```

এখানে T অবজেক্ট তৈরি করার সময় একটি রিয়েল অবজেক্ট অর্থাৎ আমরা যে অবজেক্ট প্যারিমিটারাইড করবো তা দ্বারা প্রতিস্থাপিত(replaced) হবে ।

আমরা জানি যে জাভা একটি স্ট্যাটিক টাইপ অর্থাৎ টাইপ সেইফ ল্যাংগুয়েজ। অর্থাৎ জাভা কোড কম্পাইল করার সময় এর টাইপ ইনফরমেশন ঠিক ঠাক আছে কিনা তা চেক করে নেয়।

অর্থাৎ -

```
        Generic<Integer> iObj;
```

এখানে `iObj` একটি ইন্টিজার প্যারমিটাইরজড অবজেক্ট রেফারেন্স ।

```
    iObj = new Generic<Double>(88.0); // Error!
`
```

এখন অবজেক্ট তৈরি করার সময় যদি ডাবল প্যারমিটাইরজড করি এবং `iObj` তে এসাইন করি, তাহলে

```
Error:(24, 16) java: incompatible types: Generic<java.lang.Double> cannot be converted to Generic<java.lang.Integer>
`
```

কম্পাইল করার সময় উপরের ইররটি দেখতে পাবো।

জেনেরিকস শুধুমাত্র অজজেক্ট নিয়ে কাজ করে-

আমারা জানি যে, জাভা দুই ধরণের টাইপ সাপোর্ট করে- `PrimitiveType` এবং `ReferenceType` । জেনেরিকস শুধুমাত্র ReferenceType অর্থাৎ শুধু মাত্র অবজেক্ট নিয়ে কাজ রে।

তাই-

```
     Generic<int> intObj = new Generic<int>(50);
`
```

এই স্ট্যাটমেন্ট টি ভ্যালিড নয়। অর্থাৎ প্রিমিটিভ টাই এর ক্ষেত্রে জেনেরিকস কাজ করবে না।

জেনেরিক ক্লাস এর সিনট্যাক্স-

```
class class-name<type-param-list > {}
`
```

জেনেরিক ক্লাস ইনসটেনসিয়েট করার সিনটেক্স-

```
class-name<type-arg-list > var-name = new class-name<type-arg-list >(cons-arg-list);
`
```

আমরা চাইলে একাধিক জেনেরিক টাইপ প্যারমিটাইরজড করতে পারি।

এবার তাহলে দুটি টাইপ প্যারামিটার নিয়ে একটি উদাহরণ দেখা যাক-

```
      public class Tuple<X, Y> {
          private X x;
          private Y y;

          public Tuple(X x, Y y) {
              this.x = x;
              this.y = y;
          }

          public X getX() {
              return x;
          }

          public Y getY() {
              return y;
          }

          public void showTypes() {
              System.out.println("Type of T is " +
                      x.getClass().getName() + " and Value: " + x);
              System.out.println("Type of V is " +
                      y.getClass().getName() + " and Value: " + y);
          }

          public static void main(String[] args) {
              Tuple<String, String> tuple = new Tuple<String, String>("Hello", "world");
              tuple.showTypes();

              Tuple<String, Integer> person = new Tuple<>("Rahim", 45);
              person.showTypes();
          }
       }
```

এই প্রোগ্রামটি রান করলে নিচের আউটপুট-টি পাওয়া যাবে –

> Type of T is java.lang.String and Value: Hello Type of V is java.lang.String and Value: world Type of T is java.lang.String and Value: Rahim Type of V is java.lang.Integer and Value: 45

একটি টাপলের মধ্যে আমরা চাইলে আরেকটি টাপল রাখে পারি - নিচের উদাহরণটি চমৎকার-

```
    Tuple<String, Tuple<Integer, Integer>> tupleInsideTuple = new Tuple<String, Tuple<Integer, Integer>>("Tuple", new T
```

তবে আমরা যদি জাভা ৭ অথবা ৮ ব্যবহার করি তাহলে উপরের লাইনটি সংক্ষিপ্তভাবে লিখতে পারি –

```
        Tuple<String, Tuple<Integer, Integer>> tupleInsideTuple = new Tuple<>("Tuple", new Tuple<>(45, 89));
`
```

`<>` এ একটি নতুন অপারেটর সংযুক্ত হয়েছে যাকে বলা হয় ডায়মন্ড অপারেটর। এটি ব্যবহার করে আমরা জেনেরকস এ verbosity কিছুটা কমানো যায়। অর্থাৎ

```
        Map<String, List<String>> anagrams = new HashMap<String, List<String>>();
```

এই স্ট্যাটমেন্ট-টি অনেকটাই বড়। এটি আমরা এভাবে লিখতে পারি –

```
        Map<String, List<String>> anagrams = new HashMap<>();
```

অর্থাৎ জেনেরিকস লেখার সময় বাম পাশে টাইপ প্যারামিটার ইনফরমেশন গুলো লিখলে ডান পাশে লিখতে হয় না । এটি অটোম্যাটিক্যালী ইনফার করতে পারে।

Bounded Types

আমরা উপরে দুটি উদাহরণ দেখেছি যেগুলোতে আমরা যে কোন ধরণের টাইপ প্যারামিটারাইউজড করতে পারি। কিন্তু কখনো কখনো আমাদের টাইপ restrict করতে হয়। যেমন- আমরা একটি জেনেরিক ক্লাস লিখতে চাই যা কিনা একটি এরে-তে রাখা কতগুলো নাম্বার-এর গড়(average) রিটার্ন করবে এবং আমরা চাই, এই এরে তে যে কোন ধরণের নাম্বার থাকতে পারে, যেমন- ইন্টিজার, ফ্লোটিং পয়েন্ট, ডাবল ইত্যাদি। আমরা টাইপ প্যারামিটার দিয়ে বলে দিতে চাই কখন কোনটা থাকবে। উদারহরণ দেখা যাক-

```
       public class Stats<T> {
           T[] nums;

           public Stats(T[] nums) {
               this.nums = nums;
           }

           // Return type double in all cases.
           double average() {
               double sum = 0.0;
               for (T num : nums) {
                   sum += num.doubleValue(); // Error!!!
               }

               return sum / nums.length;
           }
       }
```

এভারেজ ক্যালকুলেট করার জন্য আমাদের এভারেজ মেথড সবসময় এরে থেকে ডাবল ভ্যালু এক্সেপেক্ট করে। কিন্তু আমাদের এরে-এর টাইপ যেহেতু যে কোন রকম হতে পারে, সুতরাং সব অবজেক্ট থেকে ডাবল ভ্যালু পাওয়ার উপায় নেই।

ইনফ্যাক্ট এই ক্লাসটি কিন্তু কম্পাইল হবে না।

এই ক্লাসটিতে আমরা একটি restriction এড করতে পারি যাতে করে এই টাইপ প্যারামিটার শুধুমাত্র নাম্বার(ইন্টিজার, ফ্লোটিং পয়েন্ট,ডাবল) হবে, নতুবা এটি কাজ করবে না।

আমরা জানি যে সব নিউমেরিক অবজেক্ট গুলোর সুপার ক্লাস হচ্ছে `Number`. এবং `Number` এ `doubleValue()` মেথড ডিফাইন করা আছে। সুতরাং আমাদের ক্লাসটিকে একটু পরিবর্তন করি।

```
     public class Stats<T extends Number> {
         T[] nums;

         public Stats(T[] nums) {
             this.nums = nums;
         }

         // Return type double in all cases.
         double average() {
             double sum = 0.0;
             for (T num : nums) {
```

```
                sum += num.doubleValue(); // Error!!!
            }

            return sum / nums.length;
        }
    }
```

একটু লক্ষ্য করুন-

```
    public class Stats<T extends Number>{
    }
```

আমরা ক্লাস ডেফিনেশনে আমাদের টাইপ প্লেসহোল্ডার `T` নাম্বারকে extend করে। এটি আমাদের টাইপ প্যারামিটার পাস করতে restrict করে । অর্থাৎ আমরা শুধু মাত্র সেসব টাইপ পাস করতে পারবো যারা `Number` এর সাব টাইপ।

সুতরাং আমাদের এই `Stats` ক্লাস এখন `Integer`, `Double`, `Float`, `Long`, `Short`, `BigInteger`, `BigDecimal`, `Byte` ইত্যাদি অবজেক্ট এর জন্যে কাজ করবে।

সুতরাং দেখা যাচ্ছে যে, জেনেরিকস এর সুবিধা ব্যবহার করে আমরা এই স্ট্যাট ক্লাসটি আলাদা আলাদা করে অনেকগুলো না লিখে একটি দিয়েই কাজ করে ফেলা সম্ভব হল।

Wildcard Arguments

নিচের উদাহরণটি লক্ষ্য করি-

```
        ArrayList<Object> lst = new ArrayList<String>();
```

এটি যদি কম্পাইল করতে চেষ্টা করি, তাহলে কম্পাইলার incompatible types ইরর দেবে। কিন্তু আমরা জানি যে, সকল অবজেক্ট এর সুপার ক্লাস `Object` । তাছাড়া আমরা polymorphism থেকে জানি যে
আমরা সাব ক্লাসের রেফারেন্স কে সুপার ক্লাসের রেফারেন্স এ এসাইন করতে পারি। সুতরাং উপরের স্ট্যাটমেন্ট-টি কাজ করার কথা।

নিচের উদাহরণ দুটি লক্ষ্য করি -

```
    List<String> strLst = new ArrayList<String>();   // 1
    List<Object> objLst = strList;                   // 2 - Compilation Error
`
```

২ নাম্বার লাইনটি কাজ করছে না । যদিও বা এটি কাজ করে এবং আর্বিট্রারি কোন একটি অবজেক্ট যদি `objLst` এড করা হয় তাহলে কিন্তু `strList` করাপ্টেড হয়ে যাবে এবং সেটি আর স্ট্রিং থাকবে না।

ধরা যাক, আমরা একটা print মেথড লিখতে চাই যা কিনা একটি লিস্ট এর ইলিমেন্ট গুলো প্রিন্ট করে।

```
    public static void print(List<Object> lst) {  // accept List of Objects only,
            // not List of subclasses of object
            for (Object o : lst) {
                System.out.println(o);
            }
    }
```

এটি কিন্তু শুধুমাত্র `List<Object>` একসেপ্ট করবে, `List<String>` অথবা `List<Integer>` করবে না।

উদাহরণ-

```
    public static void main(String[] args) {
          List<Object> objLst = new ArrayList<Object>();
          objLst.add(new Integer(55));
          printList(objLst);   // matches
```

```
        List<String> strLst = new ArrayList<String>();
        strLst.add("one");
        printList(strLst);  // compilation error
    }
```

এই সমস্যা দূর করার জন্যে জাভাতে একটি একটি অপারেটর ব্যবহার করা হয় – যার নাম wildcard (?) ।

আমরা যদি আমাদের `print()` মেথডটি নিচের মতো করে লিখি, তাহলে কিন্তু আমাদের সমস্যা দূর হয়ে যাবে ।

```
public static void print(List<?> lst) {  // accept List of Objects only,
        // not List of subclasses of object
        for (Object o : lst) {
            System.out.println(o);
        }
}
```

`List<?> lst` এর মানে হচ্ছে আমরা এর টাইপ আমাদের জানা নেই, এটি যে কোন টাইপ হতে পারে । যেহেতু সব টাইপ এর সুপার ক্লাস Object সুতরাং এটি যেকোন টাইপ এর জন্যে কাজ করবে ।

Bounded Types এর মতো আমরা Wildcard Arguments কেও Bounded করে ফেলতে পারি ।

উদাহরণ -

```
    public static void process(List<? extends Foo> list) { /* ... */ }
```

এটি শুধু মাত্রে `Foo` এর সাব ক্লাস গুলো কে প্রসেস করতে পারবে । একে Upper Bounded Wildcards বলে ।

আমরা যদি এমন কোন মেথড লিখতে চাই যা শুধু মাত্র Integer, Number, and Object প্রসেস করবে অর্থাৎ Integerএবং এর সুপার ক্লাস প্রসেস করবে তাহলে -

```
    public static void addNumbers(List<? super Integer> list) {
    }
```

একে Lower Bounded Wildcards বলে ।

Generic Methods

আমরা মূলত এতোক্ষণ জেনেরিক ক্লাস নিয়ে কথা বলেছি । আমরা একটি ক্লাসকে জেনেরিক না করে শুধুমাত্রে এর একটি বা একাধিক মেথড কে জেনেরিক করে লিখতে পারি ।

উদহরণ-

```
public class Util {
    // Generic static method
    public static <K, V> boolean compare(Pair<K, V> p1, Pair<K, V> p2) {
        return p1.getKey().equals(p2.getKey()) &&
               p1.getValue().equals(p2.getValue());
    }
}
```

এটি একটি জেনেরিকম মেথড ।

জেনেরিক মেথড-এ রিটার্নটাইপ এর আগে টাইপ-প্রেস হোল্ডার `<>` লিখতে হয় ।

আমরা এবার চেষ্টা করবো কিভাবে আমরা একটি জেনেরিক সিংগলি লিংকলিস্ট লিখতে পারি --

```java
/**
 * @author Bazlur Rahman Rokon
 * @date 2/4/15.
 */
public class SinglyLinkedList<Type> {
    private long size;

    private Node<Type> head;
    private Node<Type> tail;

    public void addFirst(Type value) {
        addFirst(new Node<>(value));
    }

    public void addLast(Type value) {
        addLast(new Node<>(value));
    }

    private void addLast(Node<Type> node) {
        if (size == 0) {
            head = node;
        } else {
            tail.setNext(node);
        }
        tail = node;
        size++;
    }

    public void addFirst(Node<Type> node) {
        Node<Type> temp = head;
        head = node;
        head.setNext(temp);

        size++;

        if (size == 1) {
            tail = head;
        }
    }

    public Node<Type> getHead() {
        return head;
    }

    public Node<Type> getTail() {
        return tail;
    }

    public void removeFirst() {
        if (size != 0) {
            head = head.getNext();
            size--;
        }

        if (size == 0) {
            tail = null;
        }
    }

    public void removeLast() {
        if (size != 0) {
            if (size == 1) {
                head = null;
                tail = null;
            } else {
                Node<Type> current = head;

                while (current.getNext() != tail) {
                    current = current.getNext();
                }
                current.setNext(null);
                tail = current;

            }
            size--;
        }
    }

    public Type getFirst() {
```

```
        return getHead().getValue();
    }

    // four scenario
    // 1. empty list-  do nothing
    // 2. single node : ( previous is null)
    // 3. Many nodes
    //      a. node to remove is first node
    //      b. node to remove is the middle or last

    public boolean remove(Type type) {
        Node<Type> prev = null;
        Node<Type> current = head;

        while (current != null) {
            if (current.getValue().equals(type)) {
                if (prev != null) {

                    // just skip the current node. it works fine
                    prev.setNext(current.getNext());

                    if (current.getNext() == null) {
                        tail = prev;
                    }

                    size--;
                } else {
                    removeFirst();
                }

                return true;
            }

            prev = current;
            current = current.getNext();
        }

        return false;
    }


    public long getSize() {

        return size;
    }

    public void print() {
        System.out.print("Total elements : " + size + " -> ");
        Node node = head;
        while (node != null) {
            System.out.print(node.getValue().toString() + " ,");
            node = node.getNext();
        }
        System.out.println();
    }

    public void clear() {
        for (Node<Type> x = head; x != null; ) {
            Node<Type> next = x.next;
            x.next = null;
            x.value = null;
            x = next;
        }

        head = tail = null;
        size = 0;
    }


    private class Node<Type> {
        private Type value;
        private Node<Type> next;

        public Node(Type value) {
            this.value = value;
        }

        public Type getValue() {
            return value;
        }

        public void setValue(Type value) {
```

```
            this.value = value;
        }

        public Node<Type> getNext() {
            return next;
        }

        public void setNext(Node<Type> next) {
            this.next = next;
        }
    }
}
```

এবার আমরা এটিকে রান করে দেখি-

```
/**
 * @author Bazlur Rahman Rokon
 * @date 2/4/15.
 */
public class LinkedListDemo {
    public static void main(String[] args) {
        SinglyLinkedList<Integer> integers = new SinglyLinkedList<>();
        integers.addFirst(4);
        integers.addFirst(3);
        integers.addFirst(2);
        integers.addFirst(1);

        integers.print();

        System.out.println("Remove first and last elements..");
        integers.removeFirst();
        integers.removeLast();
        integers.print();

        System.out.println("add elements at last ");
        integers.addLast(5);
        integers.addLast(6);
        integers.addLast(7);
        integers.print();

        SinglyLinkedList<String> stringLinkedList = new SinglyLinkedList<>();
        stringLinkedList.addFirst("abcd");
        stringLinkedList.addFirst("efgh");
        stringLinkedList.addFirst("ijkl");
        stringLinkedList.addFirst("mnop");
        stringLinkedList.addFirst("qrst");
        stringLinkedList.print();
    }
}
```

Output:

> Total elements : 4 - 1 ,2 ,3 ,4 , Remove first and last elements.. Total elements : 2 - 2 ,3 , add elements at last Total elements : 5 - 2 ,3 ,5 ,6 ,7 , Total elements : 5 - qrst ,mnop ,ijkl ,efgh ,abcd ,

পাঠ ৯: জাভা আই/ও

- স্ট্রিম
- বাইট স্ট্রিম
- ক্যারেক্টার স্ট্রিম
- বাফারড স্ট্রিম
- স্ক্যানিং এবং ফরমেটিং
- ডাটা স্ট্রিম
- ইনপুট স্ট্রিম
- আউটপুট স্ট্রিম
- ফাইল
- রিডিং এ টেক্সট ফাইল
- রাইটিং এ টেক্সট ফাইল
- সারসংক্ষেপ

পাঠ ১০: জাভা এন আই/ও

- পাথ
- ক্রিয়েটিং পাথ
- রিট্রাইভিং পাথ
- ডিরেকরি এবং ট্রি
- ফাইন্ডিং ফাইল ইন ডিরেক্টরি
- ওয়াকিং থ্রু ডিরেক্টরি
- ফাইল ক্রিয়েট এবং ডিলেট করা
- দ্রুত ফাইল রিড এবং ক্রিয়েট করা
- অ্যাসিঙ্ক্রোনাস আই/ও
- সারসংক্ষেপ

পাঠ ১১: জাভা কালেকশান ফ্রেমওয়ার্ক

- জাভা কালেকশান ফ্রেমওয়ার্ক ভূমিকা
- কালেকশান ইন্টারফেইস
- লিস্ট
- সর্টেট লিস্ট
- ম্যাপ
- সর্টেট ম্যাপ
- নেভিগেবল ম্যাপ
- সেট
- সর্টেট সেট
- নেভিগেবল সেট
- Queue এবং Deque
- স্ট্যাক
- hashCode() এবং equals()
- সারসংক্ষেপ

পাঠ ১২: জাভা জেডিবিসি

- জেডিবিসি ভূমিকা
- জেডিবিস ড্রাইভার এবং টাইপস
- কানেকশান
- কুয়েরি
- রেজাল্টসেট
- প্রিপেয়ার স্ট্যাটমেন্ট
- ট্রানসেকশান
- সারসংক্ষেপ

পাঠ ১৩: জাভা লগিং

- সাধারণ ব্যবহার
- লগার
- লাগার হাইআরকি
- লগ লেভেলস
- ফরমেটারস
- ফিল্টারস
- কনফিগারেশান
- সারসংক্ষেপ

পাঠ ১৪: ডিবাগিং

- ডিবাগিং ফ্লো
- ডিবাগার দিয়ে ডিবাগিং
- ব্রেক পয়েন্ট এবং ভ্যারিয়েবলস
- স্কোপস এবং স্টেপস
- ডিবাগিং টিপস
- সারসংক্ষেপ

পাঠ ১৫: গ্রাফিক্যাল ইউজার ইন্টারফেইস

- সুইং
- কনটেইনার, কম্পোনেন্ট, ইভেন্ট, লিসেনার এবং লেআউট
- কোডিং উদাহরণ এবং ব্যাখ্যা
- সারসংক্ষেপ

পাঠ-১৬: থ্রেড

- থ্রেড কি
- থ্রেড কনস্ট্রাকশন
- রানেবল ইন্টারফেস
- থ্রেড মেথড
- থ্রেড ইন্টারেপশান
- থ্রেড স্টপ
- থ্রেড স্কেজিওলিং
- থ্রেড সেইফটি
- থ্রেড পুল
- সারসংক্ষেপ

পাঠ ১৭: নেটওয়ার্কিং

- সকেট
- ক্লায়েন্ট/সার্ভার
- টিসিপি
- ইউডিপি
- পোর্ট
- ইউআরএল
- একটি চ্যাট প্রোগ্রাম কোডিং উদাহরণ
- সারসংক্ষেপ

পাঠ ১৮: জাভা কনকারেন্সি

- ভূমিকা
- বেনিফিট
- কস্ট
- রেস কন্ডিশান এবং ক্রিটিকাল সেকশান
- থ্রেড সেইফটি এবং শেয়ার্ড রিসোর্স
- থ্রেড সেইফটি এবং ইমুট্যাবিলিটি
- সিনক্রোনাইজেশান ব্লক
- ডেড লক এবং প্রভেনশান
- রিড/রাইট লকস
- সেমাফোর
- ব্লকিং কিও
- থ্রেড পোল
- কন্ট্রোলিং এক্সিকিউশান
- মডেলিং টাস্ক
- ScheduledThreadPoolExecutor
- ফর্ক/জয়েন ফ্রেমওয়ার্ক
- একটি সিম্পল ফর্ক/জয়েন উদাহরণ
- ফর্কজয়েনটাস্ক এবং ওয়ার্ক স্টিলিং
- Parallelizing problems
- জাভা মেমরী মডেল
- সারসংক্ষেপ

পাঠ ১৯: ক্লাস ফাইল এবং বাইটকোড

- ক্লাসলোডিং এবং ক্লাস অবজেক্ট
- MethodHandle
- MethodType
- Looking up method handles
- Examining class files
- বাইটকোড
- disassembling a class
- রানটাইম ইনভারনমেন্ট
- অপকোড
- লোড অপকোড
- অ্যারিথম্যাটিক অপকোড
- Execution control opcodes
- Invocation opcodes
- Platform operation opcodes
- উদাহরণ—string concatenation
- Invokedynamic
- ইনভোটডাইনামিক কি এবং কিভাবে কাজ করে
- উদাহরণ

পাঠ ২০: Understanding performance tuning

- টারমিনলজি
- Latency
- Throughput
- Utilization
- Efficiency
- Capacity
- Scalability
- Degradation
- প্রাগমেটিং এপ্রোচ
- মুরস-ল
- মেমরী ল্যাটেন্সি হাইআরকি
- কেন জাভা পারফরমেন্স টিউনিং কেন কঠিন
- হার্ডওয়্যার ক্লকস
- কেস স্টাডি
- গারবেজ কালেক্টর
- Mark and sweep
- Jmap
- JVM প্যারামিটার
- রিডিং জিসি লগস
- ভিজুয়ালভিএম
- এসকেপ এনালাইসিস
- Concurrent Mark-Sweep
- G1—Java’s new collector
- হটস্পট দিয়ে জিট(JIT) কম্পাইলেশন
- Inlining methods
- Dynamic compilation and monomorphic calls
- Reading the compilation logs

পাঠ ২১: মডার্ন জাভা ইউজেস

- রেপিড ওয়েব ডেভেলপমেন্ট
- জাভা ফ্রেমওয়ার্ক
- স্প্রিং
- GWT
- Struts 2
- Wicket
- Tapestry
- JSF
- Vaadin
- Play
- Plain old JSP /Servlet
- অন্যান্য জেভিএম ল্যাংগুয়েজ
- গ্রেইলস
- ক্লোজার
- স্কেলা
- আরলেং
- সারসংক্ষেপ