Código da turma: 2022.2693.001

**Datas importantes**

*Aula Magna* – On-Line

Data do Evento – 11/05

*V Ciclo de Palestras*

Data do Evento – 10/08

[a confirmar]

*Férias Previstas*

18/07/2022 a 29/07/2022

*Formatura*

Data do Evento – 19 a 20 de outubro

**Metodologias ágeis**

**Frameworks ágeis (Scrum e Kanban)**

**Git e Github**

Metodologias tradicionais seguem uma sequência. Nelas, o planejamento define toda a ordem do desenvolvimento do projeto, seguindo uma abordagem linear, com passo a passo, prazo, orçamento, execução e entrega. Possuem etapas mais rígidas e controladas, por terem derivado de modelos industriais de diversos segmentos de engenharia. Justamente por isso, são de uso comum na Engenharia Civil, onde geram bons resultados. As etapas da metodologia tradicional seguem a seguinte ordem:

Desenho do projeto;

Planejamento;

Execução em detalhes.

Por isso, também é chamada de “metodologia cascata”.

A engenharia de software passou a utilizar essa metodologia, porém isso gerou problemas. Trata-se de uma área dinâmica, onde as necessidades do cliente e as tecnologias utilizadas mudam quase diariamente. A grande quantidade de concorrentes também impacta o cenário. Por isso a necessidade de uma metodologia com mais flexibilidade: as empresas têm que ter mais liberdade de modificar o ciclo do produto. Foi a partir disso que surgiu a metodologia ágil.

Na metodologia tradicional, o que se espera é que o produto saia perfeitamente pronto dentro do prazo, com todas as funcionalidades imaginadas inicialmente. Mudanças ao decorrer do ciclo geram preocupações.

As metodologias ágeis surgiram em 2001 através de um manifesto baseado em 4 valores e 12 princípios (https://agilemanifesto.org/iso/ptbr/manifesto.html). O objetivo é quebrar o produto final em pedaços menores. A agilidade, nesse caso, não vem de rapidez, e sim dessa flexibilidade e dinamismo. O projeto começa a partir de uma visão bem ampla e genérica, sendo quebrado em versões ao longo do ciclo. Muitas empresas passaram a utilizar essa metodologia.

Além disso, existem os métodos que implementam esses valores e princípios. São chamados de frameworks. É como se eles fossem mais específicos e viessem “embutidos” com a metodologia ágil, auxiliando na realização de projetos de empresas que possuem sua própria gestão.

Os principais frameworks são:

Scrum: estrutura que auxilia na organização e gerenciamento de trabalhos complexos. É simples: quanto mais desconhecemos os requisitos e tecnologias sendo utilizados, temos um cenário caótico. É aí que o Scrum se encaixa. Ele ajuda as equipes a trabalharem juntas. Possui três pilares:

Transparência dos processos, requisitos de entrega e status;

Inspeção de tudo o que está sendo feito;

Adaptação, tanto do processo, quanto do produto às mudanças.

Além dos três pilares, existem as práticas fundamentais. Primeiro, os três papéis básicos:

Scrum master: é o responsável por ajudar a todos os envolvidos a entender e abraçar os valores, princípios e práticas do Scrum. Ele deve conhecer muito bem o Scrum. Seu papel é agir como um coach, executando a liderança do processo e ajudando a equipe a desenvolver sua própria abordagem do Scrum. Também tem papel de facilitador, não sendo chefe de ninguém;

Product owner: ponto central com liderança sobre o produto. Único responsável por decidir quais recursos e funcionalidades serão construídos e qual a ordem em que devem ser feitos. É sua responsabilidade manter e comunicar a todos os outros participantes uma visão clara do que a equipe deverá buscar alcançar no projeto. É ele que prioriza os itens do product backlog;

Dev team: são as pessoas que de fato irão construir o projeto. No Scrum, aquele que decide como fazer as coisas é o time. A ideia principal é que a equipe se auto organize para determinar a melhor maneira de realizar o trabalho para atingir a meta estabelecida pelo product owner.

Em seguida, os eventos básicos: planejamento de sprints; execução de sprint; reuniões diárias; revisão do sprint e retrospectiva do sprint.

E, por fim, os artefatos gerados:

Product backlog: lista de funcionalidades necessárias do produto;

Sprint backlog: lista de atividades que precisam ser feitas durante uma sprint;

Incrementos/Entrega.

A dinâmica do Scrum começa com a visão produto, cujo responsável por provê-la é o product owner. O importante é que ele diga o que ele quer e onde quer chegar. Em seguida, deve-se desmembrar essa visão em todas as funcionalidades que são necessárias. O Scrum master, atuando como coach, auxilia o product owner nessa tarefa.

Estas funcionalidades são ordenadas por prioridade (aquilo que agrega mais valor ao negócio). Essa priorização é responsabilidade do product owner. Para fazer isso, não existe um template específico. O importante é ter, de alguma forma, uma lista de requisitos priorizados.

O projeto é planejado em sprints, que são períodos de tempos onde alguns itens selecionados do product backlog serão construídos e entregues. Para planejar os sprints, devemos obedecer a uma outra regra básica do Scrum, que são os eventos de duração fixa, também chamados de Time Box. O ideal é que todos os sprints tenham uma duração fixa e que tenham a mesma duração. Normalmente, é de 2 a 4 semanas.

Antes de cada sprint começar, é feita uma reunião de planejamento do sprint, também chamada de sprint planning, onde é criado o backlog da sprint. Com base na capacidade e velocidade da equipe, é definido quantas funcionalidades podem ser completamente construídas no tempo do sprint.

Essa dinâmica funciona da seguinte maneira: primeiramente, no backlog priorizado, são selecionadas as funcionalidades que serão feitas durante a sprint. Depois do término do sprint, é esperado que um incremento do produto seja entregue (uma parte funcional do sistema). É importante notar que os próximos itens a serem escolhidos não serão escolhidos de forma aleatória, e sim, seguindo a ordem de importância definida pelo product owner.

Conforme os incrementos do produto forem entregues, o product owner deverá verificar necessidades de mudanças. Essas mudanças devem ser inseridas também no backlog em suas devidas prioridades.

Esse processo será repetido até que todo o backlog seja construído e o produto final contemple todas as mudanças solicitadas.

Todo dia é feita uma reunião de 15 minutos, onde cada membro do time deve responder a três perguntas básicas: O que eu fiz ontem que ajudou o time a atingir a meta do sprint? O que eu vou fazer hoje para ajudar o time a atingir a meta do sprint? Existe algum impedimento que não permita a mim ou ao time atingir a meta do sprint?

Ao responder essas três questões, é possível visualizar, de uma maneira geral, como está progredindo o trabalho da sprint.

Uma ferramenta muito simples que não faz parte do Scrum, mas que é usado em 90% dos projetos, é o Burndown Chart. É um gráfico que funciona relacionando os itens a fazer com o tempo que irá demorar para fazer. Ela permite visualizar quais tarefas estão adiantas, quais estão em atraso e se a equipe pode fazer algo para ajudar a acelerar o processo.

Outra ferramenta importante é o Kanban Board: framework que auxilia no controle do projeto. É um quadro/mapa daquilo que está acontecendo na empresa e exige a colaboração da equipe. A ideia é que se possa visualizar aqui o fluxo do trabalho que está sendo feito. Pode ser feito tanto usando softwares próprios para isso, quanto usando uma lousa ou um quadro no qual são colados post-its com cada uma das funcionalidades do backlog. Nele, são divididas as tarefas a serem feitas e aquilo que já foi concluído. Foi inventado pela Toyota nos anos 40, com o objetivo de facilitar o controle do estoque de materiais, promovendo harmonia entre almoxarifado e linha produtiva. Em 2004, David J. Anderson se baseou nesse quadro para criar o método Kanban assim como é conhecido hoje em dia. Aqui, o foco é o aumento da produtividade e a otimização dos processos.

Ao final da sprint, existem duas atividades adicionais que são fundamentais. Uma delas é chamada de sprint review. O objetivo dessa atividade é validar e adaptar o produto que está sendo construído verificar as necessidades de adaptação do produto. É verificar se o que está sendo feito está de acordo com o esperado. É a apresentação daquilo que foi feito no sprint. É aqui que surgem as mudanças. É aqui que o product backlog é atualizado.

Outra atividade é a retrospectiva, que tem como objetivo verificar necessidades de adaptação no processo. É aqui que será visto o que foi feito, se foi positivo e o que foi feito que foi negativo e que deve ser melhorado ou deixado de ser feito.

Tudo começa com a visão ou esboço inicial, isso se desdobra em product backlog. Esse backlog sofre grumming, que é a sua priorização ou estimativa de tamanhos. Depois, durante o planejamento do sprint, cria-se o sprint backlog (que é a lista de necessidades que serão criadas durante as sprints). Cada sprint tem entre duas e quatro semanas e todos os dias, no mesmo lugar e horário, ocorre o daily scrum com suas três perguntas básicas e, ao final do sprint, temos um produto ou funcionalidade concluído.

Git e Github estão inseridos nesse contexto justamente por auxiliarem na gestão de projetos. São repositórios que auxiliam o trabalho em equipe.

Git, o primeiro, é um sistema de controle gratuito (open-source). Ele é utilizado para a criação de um histórico de alterações em código-fonte de projetos de desenvolvimento de software. Foi desenvolvido pelo criador do sistema operacional Linux. Ajuda no gerenciamento e organização dos códigos-fontes. A importância está em poder trabalhar em equipe, podendo fazer o compartilhamento do código entre si. Assim, podemos saber quais foram as alterações, quem fez cada uma e baixar essas mudanças. Além disso, auxilia no armazenamento e na recuperação do código.

**Conceitos importantes dentro do Git:**

Repositório: local físico onde os arquivos e suas cópias serão armazenados. O repositório pode ser local ou remoto (Github), podendo salvar não apenas arquivos de texto, mas também imagens, áudios e outros elementos relacionados ao projeto;

Branches: são os ramos, cópias do código original que podem ser manipuladas de forma livre pela pessoa que trabalha em programação, sem afetar as funcionalidades em produção no código-fonte. Isso permite que todas as alterações sejam realizadas de forma segura, sem que erros ocorram na cópia original do projeto;

Merge: diz respeito a fundir as cópias e seus arquivos modificados com o ramo principal do projeto, após a criação do branch. Isso acontece apenas no repositório local para que ajustes possam ser feitos;

Push request: envio das modificações após o merge para o repositório central, para que todas as outras pessoas que atuam no desenvolvimento possam atualizar suas cópias e revisar o código criado, verificando conflitos com seus próprios trabalhos.

Pull request: é utilizado quando outra pessoa que atua no desenvolvimento muda o ramo principal no repositório central, puxando as modificações realizadas para a sua máquina, fundindo a nova versão com o seu código local.

Fork: trata-se do comando de cópia de um repositório remoto para a máquina local, realizado sempre que vamos começar a trabalhar em um projeto que já existe. Também é usado para pegar um código público para posterior modificação e utilização.

Github: é uma plataforma totalmente on-line, onde podem ser criados nossos próprios repositórios e hospedar neles seus projetos, colaborar com softwares open-source, seguir outros programadores e interagir com códigos de terceiros (<https://github.com/>).

**Lógica de Programação**

Primeiramente, um computador é composto por:

Hardware: parte física de um computador, o conjunto de aparatos eletrônicos, peças e equipamentos que fazem o computador funcionar;

Software: parte lógica do sistema, representada por programas, apps, scripts, macros. É um conjunto de instruções que devem ser seguidas e executadas por um mecanismo, seja ele um computador ou um aparato eletromecânico.

Como ocorre o processamento de dados dentro do computador:

Entrada  Processamento  Saída

Nesse processamento muitas vezes ocorre uma interação entre as partes, através do armazenamento.

Programa: é o conjunto de instruções que descrevem uma tarefa a ser realizada por um computador. O termo pode ser uma referência ao código-fonte, escrito em alguma linguagem de programação, ou ao arquivo que contém a forma executável deste código-fonte.

Uma linguagem de programação é um conjunto de regras utilizadas para criar programas de computador. Essas regras são usadas para expressar os algoritmos de forma precisa. Há três níveis gerais de linguagens de programação:

Linguagem de máquina: lembrando que o computador corresponde basicamente um conjunto de circuitos, a sua operação é controlada através de programa s escritos numa forma bastante primitiva, baseada no sistema binário de numeração tanto para a representação dos dados quanto das operações. A esta forma de representação dos programas, é dado o nome de linguagem de máquina, em razão de ser a forma compreendida e executada pelo hardware do sistema.

Linguagem de baixo nível: embora seja a linguagem diretamente executável pelos processadores, a programação de aplicações diretamente em linguagem de máquina é impraticável. Por essa razão a linguagem de máquina de cada  
processador é acompanhada de uma versão “legível” da linguagem de máquina, chamada de Assembly;

Linguagens de alto nível: são assim denominadas por apresentarem uma sintaxe mais próxima da linguagem natural, fazendo uso de palavras reservadas extraídas do  
vocabulário corrente e permitirem a manipulação dos dados nas mais diversas formas.

Os algoritmos são escritos em uma linguagem de programação, porém, independente da linguagem utilizada e da arquitetura do sistema computacional, o código-fonte não é executável diretamente pelo processador. Isso permite apenas que o programador consiga definir o programa de uma forma legível para humanos. É necessário, então, que o código-fonte seja traduzido para o código de máquina do processador de sistema. Esse processo é realizado através de montadores e compiladores, automaticamente.

Por isso dividimos as linguagens de programação em compiladas e interpretadas. A compilação é o método que traduz todo o código-fonte em um ou mais arquivos que podem ser armazenados e executados quantas vezes se desejar, sem necessitar de nova compilação. O compilador é um software. Um exemplo de linguagem compilada é o C#.

Já a interpretação trata-se do processo em que os comandos necessários contidos no código-fonte vão sendo executados à medida em que são traduzidos, sendo necessária a interpretação toda vez que se quiser rodar o programa. Um exemplo é a linguagem Python.

Java, por sua vez, é uma linguagem tanto compilada, quanto interpretada.

O que é a lógica de programação? É a ciência de raciocínios, a técnica para colocar instruções em uma sequência lógica para que seja possível resolver qualquer problema. Extrair informações, saber como essas informações irão se comportar e escrever uma sequência lógica para atingir o objetivo. A partir disso, escrever um algoritmo e, só então, passar isso para uma linguagem de programação.

Saber analisar e resolver um problema é mais importante que decorar os comandos de uma linguagem de programação.

Essa sequência lógica é denominada na programação de algoritmo. Um algoritmo é a sequência de instruções ordenadas finita e não ambígua, que são executadas mecanicamente para executar uma tarefa. Pode se comunicar com o mundo exterior e executar suas instruções em um tempo finito. Cada linha do algoritmo deve executar uma única instrução. O algoritmo deve ser separado em fases: entrada, processamento e saída.

Uma boa técnica para se escrever um bom algoritmo é a dos cinco Qs:

1. **Q**uais são os dados de entrada necessários?
2. O **Q** devo fazer com esses dados?
3. **Q**uais são as restrições desse problema?
4. **Q**ual é o resultado esperado?
5. **Q**ual é a sequência de passos a serem feitos para chegar ao resultado esperado?

Por que é importante escrever um algoritmo eficiente e compreensível?

Como representar um algoritmo? Através de diferentes métodos: descritivo, fluxogramas, diagrama de Chapin, pseudocódigo, entre outros.

Ao descrever um algoritmo, devemos observar que algumas etapas não são tão simples por si só, são ações que precisam ser desdobradas em ações menores. A isso chamamos de ações não-primitivas. Já ações primitivas são ações que por si só são finitas e não precisam de desdobramentos.

Fluxograma é um diagrama que descreve um processo, sistema ou algoritmo de computador. Utilizam retângulos, ovais, diamantes e muitas outras formas para definir os tipos de passos, assim como setas conectoras para definir fluxo e sequência.

**Conceitos base da linguagem JAVA**

Java começou a ser desenvolvida em 1991 pela Sun Microsystems derivada do C/C++ e lançada oficialmente em 1995. Em 2009, foi comprada pela Oracle (empresa dona do banco de dados Oracle), sendo administrada e lançada por ela. Atualmente, está na versão 18, porém a mais recomendada para download é a versão 1.8 (ou versão 8).

Num sentido de plataforma, possui três versões:

JSE (Java Standard Edition): para desktop;

JEE (Java Enterprise Edition): para web;

JME (JavaMicro Edition): para mobile;

É uma linguagem executada em cima de uma máquina virtual chamada JVM (Java Virtual Machine). A função da JVM é pegar o código complexo da linguagem (chamado byte-codes) e gerar um código executável pela máquina ou sistema operacional.

É também uma linguagem compilada e interpretada. O compilador se chama javac e compila o código-fonte do Java para um código de nível intermediário chamado códigos de bytes. Esses códigos são interpretados pela JVM. Isso proporciona ao programador maior facilidade para o desenvolvimento e permite que o Java seja executado em diferentes sistemas operacionais. Por isso, dizemos que Java é uma linguagem multiplataforma (compatível com diferentes sistemas operacionais).

Java Code (.java)

javac Compiler

Byte code (.class)

JVM (Sistema operacional)

**Alguns outros termos importantes:**

JRE (Java Runtime Environment): é utilizado para executar as aplicações da plataforma Java. É composto por bibliotecas (APIs) e pela Máquina virtual Java (JVM).

JDK (Java Development Kit): é um conjunto de utilitários que permitem criar sistemas de software para a plataforma Java. É composto por compilador e bibliotecas. Contém a JRE.

Em relação à sintaxe da linguagem, temos:

**Operadores:**

São utilizados principalmente nas etapas de processamento, para a construção da lógica, realização de ações específicas e modificações sobre os dados, comparação entre variáveis, entre outros. São eles:

Atribuição (=): define um valor inicial ou sobrescreve o valor de uma variável. É usado à direita da variável a qual queremos atribuir o valor e seguido do valor em si. Exemplo: int área = 2.

Aritméticos (+, -, \*, / e %): realizam operações matemáticas como subtração, adição, multiplicação e divisão. Podem ser combinados para realizar cálculos mais complexos. Além disso, podem ser usados em conjunto com o operador de atribuição para atribuir o próprio valor da variável a ela mesma novamente, ao mesmo tempo que se realiza a operação desejada. Exemplo: área \*= 2 é o mesmo que área = area \* 2.

Incremento e decremento (++ e --): incrementa ou decrementa em 1 o valor da variável. Quando declaramos esse operador antes da variável, o incremento é realizado antes da variável ser lido para o processamento ao qual a instrução pertence. Quando declarado depois, ocorre o contrário. Exemplo: += ++desafioUm vai ter um resultado diferente de desafioDois += desafioDois++.

Igualdade e diferença (== e !=): são utilizados para comparar se duas variáveis são iguais ou diferentes, quando falamos de tipos primitivos. Ou ainda, podem ser usados para averiguar se duas instâncias diferentes apontam para o mesmo objeto.

Relacionais (<, >, <= e >=): retornam um valor booleano (verdadeiro ou falso) ao avaliar dois operandos. Definem se o operando à esquerda é menor, menor ou igual, maior ou maior ou igual ao operando à direita. Exemplo: valorA > valorB significa “valor de A é maior que o valor de B” e valorA <= valorB quer dizer “valor de A é menor ou igual ao valor de B”.

Lógicos (&& e ||): recursos que permitem criar expressões lógicas maiores a partir da junção de duas ou mais expressões. Para facilitar a leitura dessas expressões e evitar erros de lógica, é recomendado o uso dos parênteses para separar e agrupar as condições.

|  |  |
| --- | --- |
| **&&** | Utilizado quando desejamos que as duas expressões sejam verdadeiras. |
| **||** | Utilizado quando precisamos que pelo menos uma das expressões seja verdadeira. |

**Variáveis:**

Uma variável é uma estrutura que permite armazenar dados na memória durante a execução do programa, para processamento de informações. Toda variável deve ser declarada antes de ser usada, ou seja, criá-la antes de usá-la no programa. Em Java, cada variável deve ter um tipo ao ser declarada, isso porque trata-se de uma linguagem fortemente tipada.

Existem variáveis, variáveis finais, variáveis de classe e constantes. As variáveis podem ter seu valor modificado a qualquer momento, enquanto as variáveis finais e constantes recebem um valor e não podem ser alteradas.

Toda variável deve possuir um identificador, que é o seu nome. Parar criar um identificador em Java, precisamos seguir algumas regras:

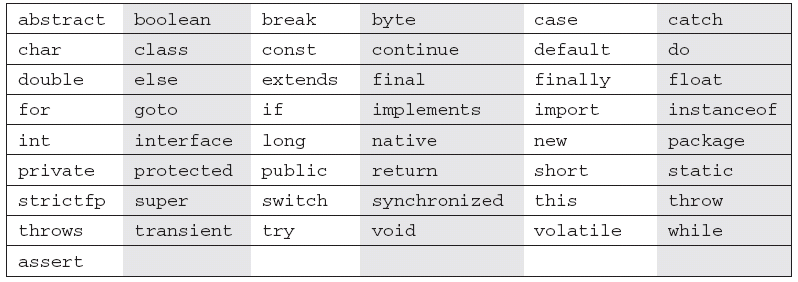
Deve conter apenas letras, \_ (underline), $ (sifrão) ou os números de 0 a 9;

Deve obrigatoriamente se iniciar por uma letra, \_ ou $;

Não podemos usar palavras-chaves da linguagem;

O nome deve ser único dentro de um escopo.

Além disso, em linguagens case sensitive (que é o caso de Java) o uso de letras minúsculas e maiúsculas é diferenciado.

 Como é declarada uma variável em Java:

Tipo identificador = valor.

A inicialização de uma variável durante sua declaração é opcional. Caso ela não seja inicializada, ela pode vir a ter um valor padrão, que depende do seu tipo declarado. Porém, é importante inicializá-la sempre que possível, mesmo que durante a execução do programa ela vá receber outros dados. É uma boa prática.

Para atribuir valor a uma variável utilizamos o operador de atribuição =.

No Java, as variáveis finais são usadas quando o valor de uma variável não vai ser alterado ao longo do algoritmo. Usa-se a palavra reservada “final” antes de declará-la. Exemplo: final String palavra = “Olá”; O valor da variável final pode ser atribuído posteriormente, não necessariamente durante a inicialização.

Constantes são estudadas junto com o conceito de orientação a objeto e classes. Assim como as variáveis finais, elas são usadas quando precisamos lidar com dados que não devem ser alterados durante a execução do programa. É declarada usando as palavras reservadas “static final”. Exemplo: public static final float PI = 3.14f;

Uma observação é que uma constante é criada na classe, ou seja, fora do método. Por esse motivo utilizamos o modificador de acesso “public”. Além disso, a constante precisa receber o seu valor no momento em que ela for declarada.

Por convenção, usamos letras maiúsculas para declarar constantes e assim distingui-las das variáveis.

**Tipos primitivos:**

Divididos em tipos primitivos e tipos por referência (tem a ver com objetos). Os tipos primitivos são boolean, byte, char, short, int, long, float e double. São internos da linguagem, ao contrário dos tipos por referência, que são objetos criados a partir de uma classe.

Uma variável primitiva armazena exatamente um valor do seu tipo declarado por vez. Quando um outro valor é atribuído a ela, seu valor anterior é substituído.

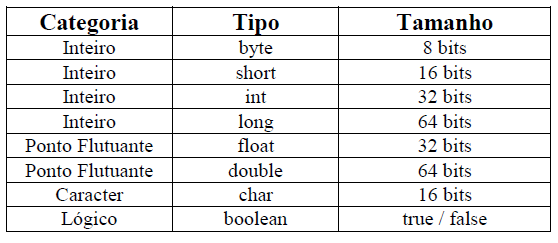
Tipos primitivos de números inteiros: byte (8 bits), short (16 bits), int (32 bits) e long (64 bits). O valor padrão de cada um deles é, respectivamente: 0, 0, 0 e 0L.

Os números de ponto flutuante são aqueles que tem a parte decimal. Os tipos primitivos de ponto flutuante em Java são: float (32 bits) e double (64 bits). As variáveis double armazenam valores com maior magnitude e precisão do que as do tipo float e devem ser preferivelmente empregados quando a precisão do valor for um fator importante.

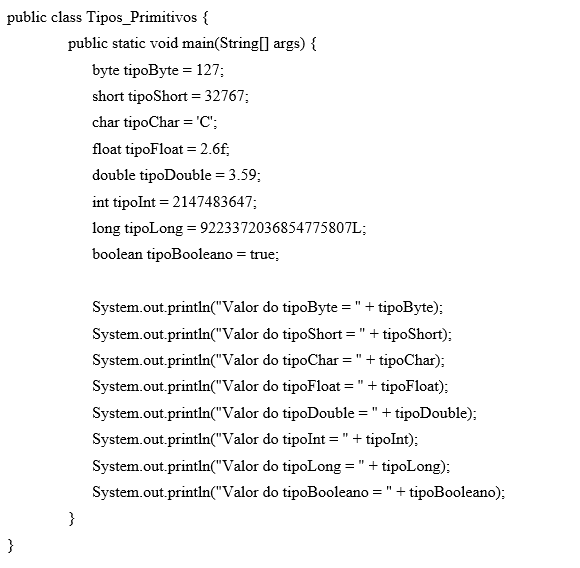
O tipo char permite armazenar um caractere Unicode, utilizando 16 bits. Seu valor mínimo é “\u0000” (0) e seu valor máximo é “\uffff” (65535). Caso seja necessário armazenar um valor maior de texto, o tipo char não deve ser usado.

O tipo boolean permite armazenar um valor lógico nos estados true ou false, ocupando apenas 1 bit de espaço. Seu valor padrão é false.

As variáveis literais são uma representação no código-fonte de um valor fixo, sendo representados diretamente no código como são, sem a necessidade de processamento.



Abaixo, um exemplo de como todos os tipos primitivos devem ser declarados na prática:



**Conversão de tipos primitivos:**

Na linguagem Java, é possível se atribuir o valor de um tipo de variável a outro tipo de variável, porém para tal é necessário que esta operação seja apontada ao compilador. A este apontamento damos o nome de *casting*.

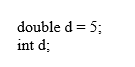
Em diversos tipos de programas, necessita-se realizar a conversão entre os diversos tipos primitivos existentes. Ao trabalhar com expressões, salvo quando todos os operandos são do mesmo tipo, é inevitável ter que considerar conversões entre um tipo de dado e outro. Em Java, existem diversas classes que realizam essa tarefa.

Existem três tipos:

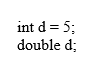
Por atribuição: ocorre quando se atribui uma variável de um determinado tipo de valor diferente do valor original. Alguns valores são incompatíveis se tentarmos fazer uma atribuição direta. A um int, por exemplo, não pode ser atribuído o valor de um double. Os seguintes trechos de código não irão compilar, pois o compilador não tem como saber qual valor entrará nesse double:

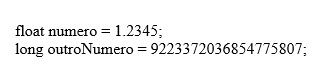




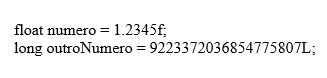


Porém, o seguinte trecho compila sem problemas, pois o tipo double pode armazenar um valor com ou sem ponto flutuante:



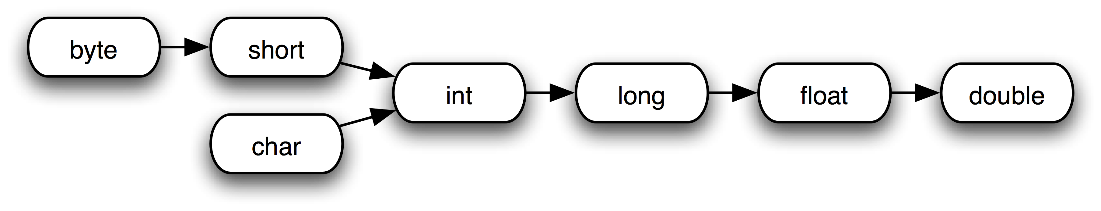


A atribuição acima vai gerar um erro, pois o tipo float não recebe valores literais. Os únicos tipos de dados que recebem valores literais são double e int, então por padrão qualquer valor literal será double ou int. Por essa razão é importante, durante a atribuição, escrever variáveis do tipo float com a letra f colada ao literal e variáveis do tipo long com a letra L:



Detalhe: o tipo de dado boolean é o único tipo primitivo que não suporta casting.

A ordem, de menor para maior, das conversões por atribuição é a seguinte:



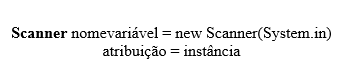
???

???

**Como utilizar o Scanner:**

Scanner é uma classe com a finalidade de facilitar a entrada de dados no modo Console.A classe Scanner tem como objetivo separar a entrada dos textos em blocos, gerando os conhecidos tokens, que são sequências de caracteres separados por delimitadores que por padrão correspondem aos espaços em branco, tabulações e mudança de linha. Com essa classe podem ser convertidos textos para tipos primitivos, sendo que esses textos podem ser considerados como objetos do tipo String, InputStream e arquivos.

Na prática, a sintaxe para declarar um Scanner é a seguinte:



**Scanner** (em negrito), nesse caso, é um tipo de dado. Quando atribuímos o novo objeto instanciado a uma variável estamos de fato criando-o.

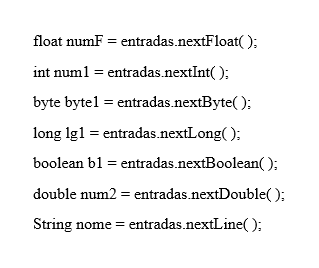
Exemplo:



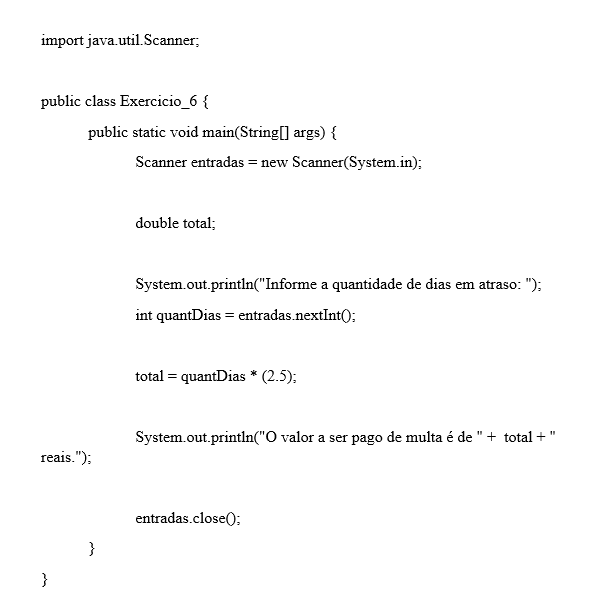
Quando invocada a classe Scanner, o compilador pedirá para fazer a seguinte importação:



O objeto System.in é o que faz a leitura do que se escreve no teclado. São invocados alguns dos métodos principais que correspondem com a assinatura que retorna um valor do tipo que foi invocado. Ou seja, para cada um dos primitivos existe uma chamada do método para retornar o valor especificado na entrada de dados, sempre seguindo o formato nextTipoDado( ). Exemplos:



Exemplo de um trecho de código utilizando Scanner:



**Estruturas de condição:**

**Operador ternário:**