# Linguagens e Ambientes de Programação (2018/2019)

## Teórica 23 (29/mai/2019)

Ambientes de programação. Estudo breve de vários casos.

# Ambientes de programação

Um **ambiente de programação** é uma coleção de ferramentas usadas no desenvolvimento de software. Um bom sistema de desenvolvimento aumenta a produtividade dos programadores e também contribui um pouco para aumentar a qualidade do software produzido.

Um ambiente minimal é tipicamente constituído por:

- um sistema de ficheiros,
- um editor de texto,
- um interpretador (ou, em alternativa, um compilador e um ligador)

Um ambiente sofisticado é tipicamente constituído por:

- um sistema de ficheiros,
- numerosas ferramentas, incluindo editor, depurador (debugger), gerador de documentação, framework para testes unitários, etc.
- essas ferramentas estão bem integradas umas com as outras;
- são oferecidas ao utilizador duas interfaces uniformes: uma interfaces gráfica e uma interface baseada em linha de comando.

# O ambiente de desenvolvimento JDK para Java

O ambiente de desenvolvimento não-visual JDK (Java Developer's Kit) consiste num conjunto de ferramentas que ajudam a **desenvolver**, **testar**, **documentar** e **executar** programas em Java. Estas ferramentas podem ser usadas na linha de comando, mas também podem ser integradas em ambientes de desenvolvimento gráficos, como por exemplo o Eclipse. O JDK é gratuitamente disponibilizado pela empresa Sun e existem versões para Windows, MacOS, Linux, etc.

## Ferramentas do JDK

| | |
|---|---|
| **javac** | Compilador |
| **java** | Executor de *programas independentes* |
| **javadoc** | Gerador de documentação |
| **appletviewer** | Executor de *applets* |
| **jdb** | Depurador |
| **javap** | Disassembler |
| **javah** | Gerador de *header files* para métodos nativos escritos em C |

**Como se compilam programas em Java?**

- Os programas em Java compilam-se por meio do comando **javac**. Por exemplo: `javac Sets.java`. O comando **javac** permite compilar tanto **programas independentes** como **applets**.
- Os ficheiros compilados pelo comando **javac** têm obrigatoriamente a extensão ".java".
- Quando cativado, o compilador cria na diretoria corrente diversos ficheiros com a extensão ".class", um por cada classe ou interface definidas nos ficheiros fonte. Os ficheiros ".class" contêm código executável, num formato padrão chamado **bytecode**.

**Como se executam programas em Java?**

- Se se tratarem de **programas independentes**, executam-se usando o comando **java**. Por exemplo: `java TestSets`. Neste exemplo, "TestSets" refere-se ao ficheiro "TestSets.class" na diretoria corrente: repare que o comando `java` não admite a escrita da extensão ".class".
- Se se tratarem de **applets**, executam-se usando o comando **appletviewer** ou então por intermédio dum browser WWW.
- A componente de software que permite executar ficheiros de de *bytecode* chama-se **Java Virtual Machine**. Está incorporada no comando **java**, no comando **appletviewer** e na maioria dos browsers WWW modernos.

**Como se documentam programas em Java?**

- O gerador de documentação, **javadoc** converte ficheiros ".java" em ficheiros HTML, visualizáveis usando qualquer browser WWW. Estes ficheiros HTML incluem documentação sobre as classes, interfaces, métodos, variáveis e exceções definidos nos ficheiros fonte e ainda o conteúdo de comentários especiais, da forma /** ... */, escritos pelo programador. A vasta documentação sobre a plataforma Java que é fornecida pela Sun foi gerada pelo comando **javadoc**.

# As ferramentas do OCaml

A distribuição do OCaml inclui uma vasto conjunto de ferramentas disponíveis no Windows, Linux e MacOS. Estas ferramentas podem ser usadas na linha de comando, mas também podem ser integradas em ambientes de desenvolvimento gráficos.

Repare que a lista de ferramentas abaixo inclui um profiler que serve para o programador descobrir quais as partes do programa onde se gasta mais tempo. Essas são as partes do programa que mais interessa optimizar.

## Ferramentas do OCaml

| | |
|---|---|
| **ocaml** | Interpretador |
| **ocamlc** | Compilador para a máquina virtual CAML |
| **ocamlopt** | Compilador de código nativo |
| **ocamlrun** | Executor da máquina abstrata CAML |
| **ocamlbrowser** | Permite navegar e inspecionar o conteúdo dos módulos de biblioteca |
| **ocamldebug** | Depurador de código fonte |
| **ocamldoc** | Gerador de documentação |
| **ocamldep** | Inspeciona um conjunto de ficheiros fonte e gera automaticamente informação de dependências para a ferramenta make |
| **ocamlcp** | Profiler, insere no código fonte a contagem de quantas vezes cada função é chamada, etc. |
| **ocamlprof** | Interpreta o output da execução de programas processados usando ocamlcp |
| **ocamllex** | Gera reconhecedores de expressões regulares diretamente a partir dessas expressões |
| **ocamlyacc** | Gera parsers diretamente a partir de gramáticas LALR(1) |
| **ocamlp4** | Processador do código fonte de programas ocaml (para pretty-print, por exemplo) |
| **ocamldumpobj** | Disassembler de ficheiros .cmo |
| **ocamlobjinfo** | Inspeciona ficheiros .cmo, .cmi, .cma |

---

# Unix e Gnu/Linux

O Unix é um sistema operativo mas também um sistema de desenvolvimento e manutenção de software. Atualmente a sua versão open-source, chamada de Gnu/Linux, é muito popular e tem sido usada no desenvolvimento de software "open-source" e de software comercial.

O Unix disponibiliza uma grande quantidade de ferramentas de desenvolvimento, mas deve ser dito que essas ferramentas foram nascendo de forma caótica e que não há uma interface consistente entre elas. Algumas ferramentas são também muito complexas, embora cumpram os seus objetivos de forma admirável, na maior parte dos casos.

## gnu tools

Os **gnu tools** são um conjunto de ferramentas que ajudam a produzir software portável, que funciona em diversas versões do Unix e noutros sistemas operativos. Estas ferramentas foram desenvolvidas e adquiriram maturidade especialmente durante os anos 1990.

| | |
|---|---|
| **autoconf** | Ferramenta para criar projetos de software portáveis e reconfiguráveis |
| **automake** | Gerador automático de makefiles |
| **libtool** | Ferramenta para criar bibliotecas de software |

## make

A utilitário **make** pode ser usado para gerir automaticamente pequenos projetos. A sua principal utilização é determinar automaticamente que partes do projeto precisam de ser recompiladas e produzir os comandos para concretizar a recompilação. O make considera as dependências entre os diversos ficheiros do projeto para saber o que é precisos fazer em cada momento.

Geralmente, cada projeto contém um ficheiro chamado **Makefile** onde se declaram as dependências entre os ficheiros, e onde se definem regras que dizem como os diversos ficheiros devem ser atualizados. Para além da recompilação, podem ser definidas outras ações, tais como apagar os ficheiros auxiliares do projeto, ou produzir o arquivo da distribuição do projeto.

O seguinte exemplo é o ficheiro Makefile que foi usado num projeto antigo de LAP. O ficheiro declara as dependências entre ficheiros e tem regras que dizem como obter uns ficheiros a partir de outros (as regras começam obrigatoriamente por um tab).

```
APP = dt
VERSION = 0.1
COMP = ocamlc
INTERFS = DTree.mli
OBJS = DTree.cmo dt.cmo
GROUP = 123_456

$(APP): $(OBJS)
	$(COMP) -o $(APP) $(INTERFS) $(OBJS)

%.cmo: %.ml
	$(COMP) $(INTERFS) -c $<

clean:
	rm -f $(APP) *.cmo *.cmi

dist: clean
	mkdir $(GROUP)
	cp *.ml *.mli $(GROUP)
	zip -r $(GROUP).zip $(GROUP)
	rm -rf $(GROUP)
```

Eis uma pequena sessão interativa, que pressupõe a existência do ficheiro `Makefile` na diretoria corrente:

```
$ ls
```

```
dt.ml  DTree.ml  DTree.mli  Makefile
$ make DTree.cmo
ocamlc DTree.mli -c DTree.ml
$ make
ocamlc DTree.mli -c dt.ml
ocamlc -o dt DTree.mli DTree.cmo dt.cmo
$ make clean
rm -f dt *.cmo *.cmi
$ make dist
rm -f dt *.cmo *.cmi
mkdir 123_456
cp *.ml *.mli 123_456
zip -r 123_456.zip 123_456
  adding: 123_456/ (stored 0%)
  adding: 123_456/DTree.ml (deflated 72%)
  adding: 123_456/dt.ml (deflated 67%)
  adding: 123_456/DTree.mli (deflated 65%)
rm -rf 123_456
$ ls
123_456.zip  dt.ml  DTree.ml  DTree.mli  Makefile
```

## gdb

O Gnu Debugger trabalha com linguagens implementadas nativamente e tem um funcionalidade extensíssima. Mostramos só como usar o `gdb` para descobrir qual a função onde um programa está a *rebentar*.

O seguinte programa está errado:

```
char str[128] ;

void f(void)
{
    char *pt = str ;
    int i ;
    for( i = 0 ; i < 10000 ; i++ ) <-- Provoca estoiro
        *pt++ = 'a' ;
}

void g(void) {
    f() ;
}

int main()
{
    g() ;
    return 0 ;
}
```

O `gdb` permite mostrar o conteúdo da pilha de execução no momento do estoiro, o que geralmente é muito útil para descobrir as razões do erro.

```
$ ./a
Segmentation fault (core dumped)
$ gdb ./a
GNU gdb 6.6-debian
Copyright (C) 2006 Free Software Foundation, Inc.
(gdb) run
Starting program: /media/EXTERN/amd1/z/a

Program received signal SIGSEGV, Segmentation fault.
0x0804835d in f ()                    <-- Estoiro na função f
(gdb) backtrace
#0  0x0804835d in f ()
#1  0x0804837b in g ()
#2  0x08048390 in main ()             <-- Pilha de execução no momento do estoiro
(gdb)
```

## cvs

O **cvs** (Concurrent Versions System) um sistema de apoio ao desenvolvimento de projetos de software. Tem duas funcionalidades essenciais:

- Suporta o **registo cronológico** das alterações introduzidas nos ficheiros fonte do projeto;
- Suporta a **colaboração** entre os diversos participantes de projeto, e sabe lidar com o problema da submissão de **alterações conflituantes**.

O CVS, divulgado em 1986, foi o primeiro sistema de controlo de versões a suportar um repositório partilhado por vários utilizadores.

Uma ferramenta como o CVS é essencial para a viabilidade do software livre, pois este tipo de software é tipicamente desenvolvido coletivamente por um grande número de voluntários geograficamente espalhados e não pagos. Nenhum desses voluntários teria tempo nem paciência para processar manualmente as numerosas submissões de código novo ou corrigido.

Uma ferramenta do tipo do CVS também faz falta no desenvolvimento de software proprietário, mas no caso do software livre a questão é mais evidente.

**Exemplos de software originalmente desenvolvido com a ajuda do CVS**

- Apache web server

- X window system
- GNOME desktop
- Mozilla web browser
- OpenOffice
- GCC gnu compiler collection
- Debian linux distribuition
- Anjuta IDE for rapid application development

Em 1999 foi lançado o Sourceforge, um repositório de código fonte CVS baseado na Web. Foi um dos primeiros sistemas Web a oferecer a possibilidade de gestão colaborativa de projetos de código aberto. Neste momento o sistema hospeda mais de 300 000 projetos, embora nem todos estejam ativos. O próprio sistema Sourceforge era inicialmente baseado em código aberto, embora tal não seja verdade atualmente.

**Alternativas modernas ao CVS**

O CVS tem algumas limitações aborrecidas, como por exemplo a falta de suporte para a noção de renomear uma diretoria. Atualmente há muitas alternativas ao CVS e, para iniciar projetos novos, convém evitar o CVS.

- Alternativas *open source*: há cerca de duas dezenas de alternativas, por exemplo, Subversion, Monotone, Git, Bazzar.
- Alternativas comerciais: há apenas meia dúzia, por exemplo, BitKeeper e Plastic SCM.

**Subversion** (2004) - O Subversion foi criado para resolver as insuficiências mais evidentes do CVS. Foi a primeira alternativa popular ao CVS e continua a ser muito usado.

**Git** (2005) - Mas, atualmente, o sistema mais usado é provavelmente o Git, por causa de diversas funcionalidades específicas e duma boa velocidade de resposta, mesmo em projetos grandes. O plugin para o Eclipse EGit constitui uma interface prática para usar o Git.

**GitHub** (2008) - Atualmente é o maior repositório de código fonte do mundo. Usa o Git e está acessível na Web. Tem mais de 21 milhões de projetos instalados, embora nem todos estejam ativos.

**Detalhes sobre o Git** - Todos os participantes dum projeto possuem a sua cópia local do projeto e de todo o código. Trabalham localmente sobre o código e periodicamente usam os comandos do Git para atualizar o estado do projeto local. Sempre que um participante considerar conveniente, dá o comando que incorpora no projeto central as alterações entretanto introduzidas no projeto local. Eis alguns exemplos de comandos:

```
git clone https://user@github.com/Project/project.git
git status
git add myfile
git commit
git push
```

# Conhecer melhor o Git em 15 minutos!

---

# IDEs

Um IDE, ou ambiente de desenvolvimento integrado, é um programa único dentro do qual se processa todo o desenvolvimento do software. Geralmente a interface é gráfica.

Exemplos: Eclipse, Visual Studio .Net, Anjuta, Code::Blocks.

Os IDEs mais sofisticados são estendíveis através de plugins. Isso permite que o mesmo sistema possa ser usado para trabalhar em diversas linguagens.

---

#80