# Sviluppo di Software Sicuro - S$^3$
# SPARK - Introduzione

Corso di Laurea Magistrale in
Sicurezza Informatica: Infrastrutture e Applicazioni
Università di Pisa – Polo di La Spezia
C. Montangero
Anno accademico 2009/10

## Sommario

- Scopo del linguaggio
  - High integrity -> sicurezza
- Natura del linguaggio
  - imperativo, sequenziale (per noi), derivato da Ada
- Contesto: specifica di dettaglio, validata
- Concetti base



S$^3$ 2009/10 – SPARK - Introduzione

## SCOPO DEL LINGUAGGIO

## Obiettivi

- Produrre codice con pochi difetti
  - Evitando alcuni pericolosi errori di codifica
    - http://cwe.mitre.org/top25
  - In particolare:
    - inizializzazioni improprie (improper initialization)
    - controllo improprio delle esondazioni (buffer overflow errors)

S3: SPARK - C.Montangero - Copyright 2010 4

## Obiettivi (2)

- Grazie ad analisi *statiche*
  - flusso dei dati (*data flow*)
    - difetti di inizializzazione

```
1 public class Data {
2     public int info;
3     public Data() { info = 0; }
4     public void exchange(Data y, Data z) {
5         Data t = new Data();
6         t.info = y.info;
7         y.info = z.info;
8         z.info = t.info;
```

S3: SPARK - C.Montangero - Copyright 2010 5

## Obiettivi (2)

- Esempio di difetto:

```
1 public class Data {
2     public int info;
3     public Data() { info = 0; }
4     public void exchange(Data y, Data z) {
5         Data t           ;
6         t.info = y.info;
7         y.info = z.info;
8         z.info = t.info;
```

- Anche in Java, e altri linguaggi

S3: SPARK - C.Montangero - Copyright 2010 6

## Obiettivi (3)

– Grazie ad analisi *statiche*
  • flusso delle informazioni (*information flow*)
    – mancato uso di informazioni necessarie

```
1 public class Data {
2     public int info;
3     public Data() { info = 0; }
4     public void exchange(Data y, Data z) {
5         Data t = new Data();
6         t.info = y.info;
7         y.info = z.info;
8         z.info = t.info;
```



## Obiettivi (3)

– Grazie ad analisi *statiche*
  • flusso delle informazioni (*information flow*)
    – mancato uso di informazioni necessarie

```
1 public class Data {
2     public int info;
3     public Data() { info = 0; }
4     public void exchange(Data y, Data z) {
5         Data t = new Data();
6
7         y.info = z.info;
8         z.info = t.info;
```



## Obiettivi (4)

– Grazie ad analisi *statiche*
  • flusso delle informazioni (*information flow*)
    – creazione di informazioni superflue (comandi inutili)

```
1 public class Data {
2     public int info;
3     public Data() { info = 0; }
4     public void exchange(Data y, Data z) {
5         Data t = new Data();
6         t.info = y.info;
7         y.info = z.info;
8         z.info = t.info;
```

## Obiettivi (4)

– Grazie ad analisi *statiche*
  • flusso delle informazioni (*information flow*)
    – creazione di informazioni superflue (comandi inutili)

```
1 public class Data {
2     public int info;
3     public Data() { info = 0; }
4     public void exchange(Data y, Data z) {
5         Data t = new Data();
6         t.info = y.info;
7         y.info = z.info;
8         z.info = y.info;
```

inutile!

!!!!!!



## Obiettivi (4)

- Produrre codice *senza* difetti
  – A partire da una specifica formale in Z
    • da cui derivare le annotazioni del codice
  – Grazie a strumenti di prova *semi*-automatica
    • generatore di condizioni di verifica
      – nessuna, nel caso migliore
    • scaricate *esplicitamente*, se non formalmente
- Graduazione nell'applicazione delle verifiche
  – a seconda della criticità del componente



## Obiettivi (5)

- Conservare quelli di Ada:
  – affidabilità e manutenibilità del codice
  – efficienza del codice
  – supporto al programmatore
    • pochi concetti ben integrati (modulo l'ampiezza dell'obiettivo)
    • applicabilità ampia: software embedded, scientifico e amministrativo
- Favorire le buone pratiche di Ingegneria del Software
- Buona struttura dei programmi
  – information hiding
    • interfaccia vs implementazione
      – `function, procedure, package`
      – e relativi `body`
  – lavoro in parallelo
    • compilazioni separate
      – `private`

S3 2009/10 – SPARK - Introduzione

## NATURA DEL LINGUAGGIO



## Caratteristiche ereditate da Ada

- imperativo: basato sulla nozione di stato
- fortemente tipato
  - solo frasi che combinano correttamente operatori e operandi sono lecite (accettate dal compilatore)
- con tipi astratti
  - operazioni pubbliche, implementazione privata
- modulare
  - dichiarazione statica di moduli
  - compilazioni separate



## Caratteristiche proprie

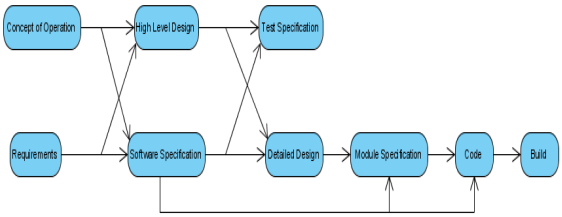
- sequenziale: un solo flusso di controllo
  - estensioni in lavorazione
- no eccezioni
  - se ce ne sono, il programma muore
    - un programma sicuro non deve sollevare eccezioni
      - pena DOS
    - un programma affidabile pure, pena danni irreparabili
- annotazioni ( dentro commenti - -# )
  - per permettere l'analisi del flusso delle informazioni
  - per permettere la prova di correttezza

S³ 2009/10 – SPARK - Introduzione

## CONTESTO: SPECIFICA DETTAGLIATA



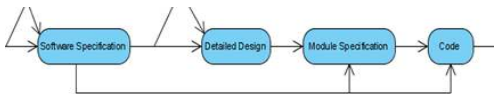
## Correctness by Construction




– Processo di base per SPARK
  • notare la presenza di test di sistema

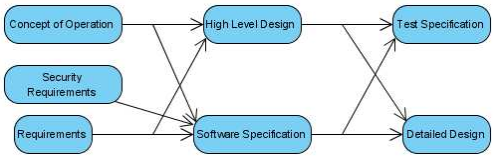


## Codifica




- La codifica è a valle di
  – Specifica dei moduli
    • Usa la specifica formale per specificare i moduli
  – Specifica formale (SW specification)
    • Viene usata per dar significato alla specifica del moduli
      – e.g ., strutture dati

## Security by Construction




- Le attività di validazione comprendono
  - prova della correttezza della specifica
  - rispetto ai requisiti di sicurezza

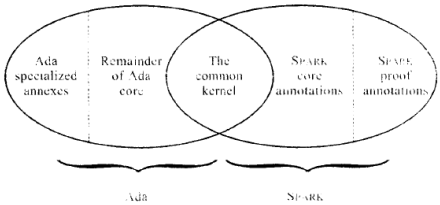


S³ 2009/10 – SPARK - Introduzione

## SPARK: CONCETTI DI BASE



## SPARK e Ada



## Tre aree

- Modello dei tipi
  - ristretto per permettere verifiche statiche
  - no puntatori
  - sottoinsieme proprio di Ada
- Flusso di controllo e dati
  - condizionali, analisi di casi, cicli
  - no salti, no eccezioni
  - sottoprogrammi: procedure e funzioni
  - annotazioni
    - per il flusso dei dati nei/dai sottoprogrammi



## Tre aree (2)

- Struttura e
  - visibilità
    - **package**
      - definiscono la macro struttura
      - con i tipi **private** controllano la visibilità
      - con la clausola **with** aprono lo scope
      - specificano interfacce, con *inizializzazioni statiche*
    - sottoprogrammi (**procedure**, **function**)
      - definiscono la microstruttura
    - Insieme: tipi di dati astratti



## Tre aree (3)

- Struttura e
  - compilazioni separate
    - (**package** | **procedure** | **function**) **body**
      - permettono di compilare in presenza delle sole interfacce
      - grazie a private non devono ricorrere all'accesso indiretto, come in Java
  - programmi
    - un programma SPARK è una libreria di
      - package utente
      - package Standard
      - un'unica procedure annotata come punto d'ingresso

```
--# main_program ;
procedure UnProgrammaSPARK
```

## Lessico

- Insieme di caratteri: Latin-1
  - Tabella A2.1 di HIS
- Identificatori
  - solite regole: inizio lettera, solo lettere e cifre
    - non distingue maiuscole e minuscole
    - _ lecito come separatore (vs CamelCase)
  - tabù:
    - predefiniti in Standard
    - parole chiave di SPARK e FDL (verifica correttezza)
    - vedi Appendice 2 di HIS



## Lessico (2)

- Commenti
  -- questo è un commento (analogo // Java)
  --# questa è un'annotazione SPARK
- Costanti numeriche
  - intere: 255 2#1111_1111# 16#FF#
  - reali: 255,0 0,255E+3
  - booleani: True, False
  - caratteri: 'A' '"' ' '
  - stringhe: "Una" "Stringa"
    "UnaStringa" & Ada.Characters.Latin_1.NUL
    (la concatenazione &, solo *statica*)



$S^3$ 2009/10 – SPARK - Introduzione

**PROSSIMO ARGOMENTO:**
**SPARK – MODELLO DEI TIPI**