

Bezpieczeństwo technologii Java

Michał Stachoń
Fundacja Wspierania Edukacji Informatycznej PROIDEA

Agenda

- Czym jest technologia Java?
- Podstawowe cechy Javy
- Potrzeba zabezpieczeń
- Polityka bezpieczeństwa
 - bezpieczeństwo języka
 - bezpieczeństwo czasu wykonania
- Ewolucja modelu bezpieczeństwa

Technologia Java

- Język programowania
 - składnia w dużej mierze oparta na języku C++
- Środowisko do tworzenia aplikacji
 - kompilator, debugger, generator dokumentacji, narzędzie do pakowania klas i inne
- Środowisko uruchomieniowe
 - Wirtualna Maszyna Javy (JVM)

Technologia Java

- Opracowana do zastosowań w otwartych i heterogenicznych środowiskach sieciowych.
 - zapewnienie bezpieczeństwa jednym z głównych celów projektantów SUN'a
- Niezależna od platformy sprzętowej i programowej
 - programy kompilowane do kodu pośredniego tzw. bytecode'u (kodu bajtowego)
 - niewielki rozmiar kodu pośredniego

Wirtualna Maszyna Javy

- Wirtualny komputer posiadający własne rejestry, stos i stertę.
- Interpretuje kod pośredni do kodu wykonywalnego dla danego systemu operacyjnego.
- JVM dostępna jest dla systemów operacyjnych:
 - *Windows, Linux x86/A64, Solaris SPARC/x86/A64*
- Dzięki standaryzacji JVM programy napisane w Javie wykonują się identycznie na każdej platformie.

Podstawowe cechy

- Język prosty
 - zrezygnowano z elementów będących często źródłem problemów np. przeciążanie operatorów
- Zorientowany obiektowo
 - tworzenie łatwych w późniejszej pielęgnacji rozbudowanych systemów informatycznych
- Jest otwartym standardem
 - pełna specyfikacja wraz z kodem źródłowym publicznie dostępna

Podstawowe cechy

- Bezpieczny
 - szczególnie w porównaniu z innymi językami
- Posiada olbrzymią bibliotekę gotowych klas
 - tworzenie GUI, klasy kolekcji, przetwarzanie XML, obsługa sieci, itp.
- Nie posiada w zasadzie nic wspólnego, poza nazwą, z JavaScript!!!

Potrzeba zabezpieczeń

- Java powstała głównie z myślą o lokalnym wykonywaniu kodu pobranego z sieci.
- Kod ten może zawierać błędy powstałe w skutek niedopatrzenia, np.
 - brak kontroli parametrów wejściowych
 - wycieki pamięci
- Celowo umieszczony niebezpieczny (złośliwy) kod

Potrzeba zabezpieczeń

- Należy jak najbardziej ograniczyć możliwość wykonania kodu mogącego spowodować:
 - uszkodzenie sprzętu, oprogramowania lub informacji na lokalnej maszynie
 - wyciek (przekazanie) informacji do niepowołanych osób
 - lokalny DoS – brak możliwości normalnego korzystania z maszyny z powodu wyczerpania zasobów systemowych

Polityka bezpieczeństwa

- Dotyczy
 - języka (bezpieczeństwo języka)
 - środowiska wykonawczego (bezpieczeństwo czasu wykonania)
- Może być bardzo restrykcyjna
- Łatwo i wysoce konfigurowalna

Bezpieczeństwo języka

- Obiektowość
- Specyfikatory dostępu
- Brak wielokrotnego dziedziczenia
- Brak arytmetyki wskaźnikowej
- Odśmiecacz pamięci
- Kontrola zakresu indeksów tablic
- Niezmienialny typ łańcuchowy

Bezpieczeństwo języka

- Bezpieczeństwo typów (rzutowania)
- Mechanizm obsługi wyjątków
- Domyślna inicjalizacja
- Logowanie i asercje
- Brak przeciążania operatorów

Obiektowość

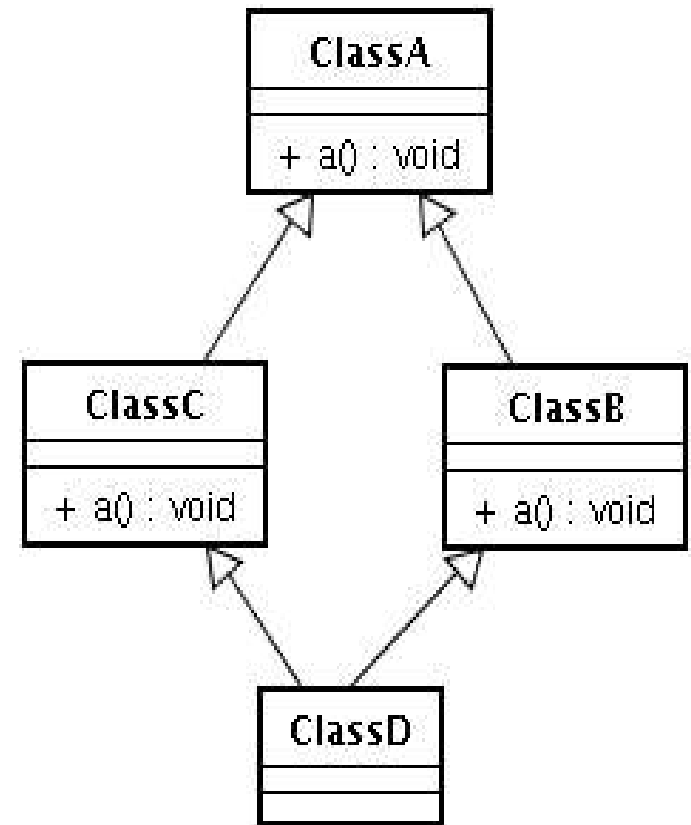
- Kod zawsze zamknięty w klasie
 - Java „wymusza” na programiście pisanie programów w sposób obiektowy
- Efektywne reużytkowanie kodu
 - mechanizm dziedziczenia i kompozycji
- Łatwość tworzenia złożonych i prostych w pielęgnacji systemów informatycznych

Specyfikatory dostępu

- Określają kto ma dostęp do danej definicji
- Pozwalają ukryć szczegóły implementacji
 - problem twórców (autorów) i klientów (użytkowników) klas
 - łatwość wprowadzania zmian
- Brak zamkniętych pakietów!
 - każda nowa klasa można być członkiem dowolnego pakietu

Brak wielokrotnego dziedziczenia

- Drzewiasta hierarchia klas
 - wyklucza powstawania tzw. struktur kratowych i ewentualne niejednoznaczności dziedziczenia
- Obiekt nie może być dwóch lub więcej typów
- Możliwość wielokrotnej implementacji interfejsów



Brak arytmetyki wskaźnikowej

- Kod pośredni odwołuje się do pamięci poprzez uchwyty symboliczne
 - odwzorowanie na adresy rzeczywiste podczas ładowania klasy
- Brak możliwości fałszowania lub zmiany wskazań uchwytów na inny obszar pamięci.
- Korzyść a nie ograniczenie programisty.

Odśmiecacz pamięci

- Nie istnieje problem wycieku pamięci
 - brak konieczności jawnego niszczenia obiektów i zwalniania pamięci po nich
- W specyficznych (bardzo :)) przypadkach zwiększenie wydajności systemu
 - w przypadku wystarczającej ilości wolnej pamięci GC może zostać nigdy nie uruchomiony

Kontrola zakresu indeksów tablicy

- Tablica jest obiektem, a nie ciągłym obszarem pamięci
 - posiada atrybut tylko do odczytu określający jej długość
 - możliwość kontroli zakresu
- Narzut obliczeniowy przy każdym odwoływaniu się do elementu tablicy

Niezmiennność typu łańcuchowego

- Łańcuch jest obiektem, a nie tablicą znaków
- W celu modyfikacji napisu należy utworzyć nowy obiekt
 - uniemożliwia praktycznie wprowadzenie do napisu kodu wykonywalnego
 - brak arytmetyki wskaźnikowej uniemożliwia jego uruchomienie
- Java dostarcza inny typ obiektu do wykonywania operacji na napisach

Bezpieczeństwo typów (rzutowania)

- Sprawdzanie przez kompilator i interpreter czy rzutowanie jest dozwolone
 - nie wszystkie błędy dają się zauważyć podczas kompilacji
- Brak kontroli nad rzutowaniem (np. C/C++) łatwo może prowadzić do nieuprawnionego dostępu do pamięci.

Mechanizm obsługi wyjątków

- Jest częścią języka i wymusza na programiści napisanie kodu obsługi sytuacji wyjątkowej
- Eliminuje przypadkowość obsługi błędów
- Pozwala oddzielić kod rozwiązujący dany problem od kodu obsługi błędu
 - do jednego typu wyjątku wystarczy jeden kod obsługi wyjątku
 - kod łatwej pisać i czytać

Domyślna inicjalizacja

- Atrybuty klas i elementy tablic inicjalizowane do określonych wartości
- Użycie niezainicjalizowanej zmiennej lokalnej zgłaszane przez kompilator jako błąd
- Pozwala uniknąć błędów polegających na przypadkowym użyciu zmiennej o bliżej nieokreślonej wartości

Logowanie

- Od wersji 1.4 biblioteka `java.util.logging` jest częścią standardowego API
- Pozwala w prosty sposób zapisywać stan przebiegu działania aplikacji
- Konfigurowalność
 - filtrowanie
 - ustalanie poziomu logowania

Mechanizm asercji

- Od wersji 1.4 Javy
- Pozwala sprawdzić czy założenia projektowe odnośnie pewnych wyrażeń są spełnione
- Działa tylko, gdy odpowiednia opcja zostanie wybrana przy uruchamianiu aplikacji

Brak przeciążania operatorów

- Projektanci Javy stwierdzili, że jest to trudne i stosunkowo często prowadzi do błędów.
- Wyjątkiem jest typ łańcuchowy (String), dla którego przeciążono operator „+” i „+=”
 - stosunkowo często używany
 - upraszcza zapis i zwiększa czytelność kodu

Bezpieczeństwo języka – podsumowanie

- Prowadzi do powstawania bardziej niezawodnych i bezpiecznych aplikacji
- Nie wszystkie problemy są rozwiązane
 - mechanizm kontroli rzutowania da się oszukać
 - brak zamkniętych pakietów
- Funkcjonalność niektórych z przedstawionych mechanizmów da się uzyskać w językach ich pozbawionych

Bezpieczeństwo czasu wykonania

- Nie można ufać, że programiści
 - będą używać oficjalnych kompilatorów
 - nie będą dokonywać zmian w kodzie pośrednim
- Odpowiedzialności za bezpieczeństwo spoczywa również na JVM
 - Class Loader
 - Code Verifier
 - Security Manager

Class Loader

- Odpowiedzialny za ładowanie kodu pośredniego
 - odrębne przestrzenie nazw dla klas lokalnych oraz dla klas pobranych z sieci
 - ustalanie „rozmieszczenia” klasy w pamięci
 - odwzorowanie nazw uchwytów symbolicznych na rzeczywiste adresy pamięci
- Klasy wczytywane dopiero, gdy są potrzebne
 - przy pierwszym ich użyciu

Code Verifier

- Sprawdza czy kod pośredni spełnia założenia specyfikacji JVM w tym m. in.:
 - czy klasa ma poprawną strukturę i jest spójna z wcześniej załadowanymi
 - nieprawidłowości zarządzania pamięcią
 - naruszenia ograniczeń dostępu
 - występowanie nielegalnych konwersji
 - parametry kodu pośredniego

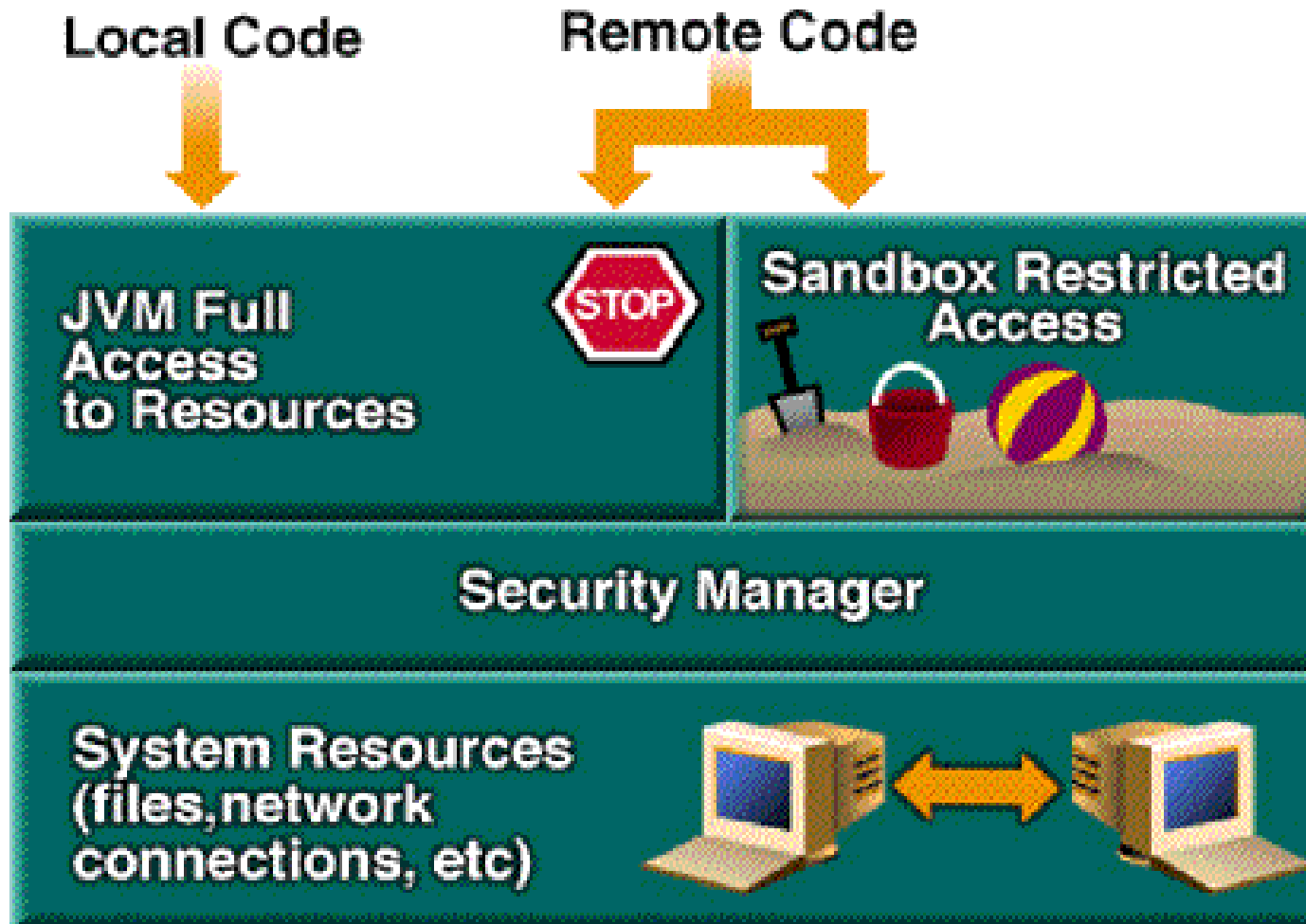
Security Manager

- Odpowiedzialny za kontrolę aplikacji w trakcie jej działania
- Zawsze odpytywany przed wykonaniem potencjalnie niebezpiecznej operacji
- Zestaw metod typu „sprawdź”
 - `checkRead(String file)`
 - `checkPermission(Permission prm, Object context)`
- Metody bezpośrednio zakodowane w API

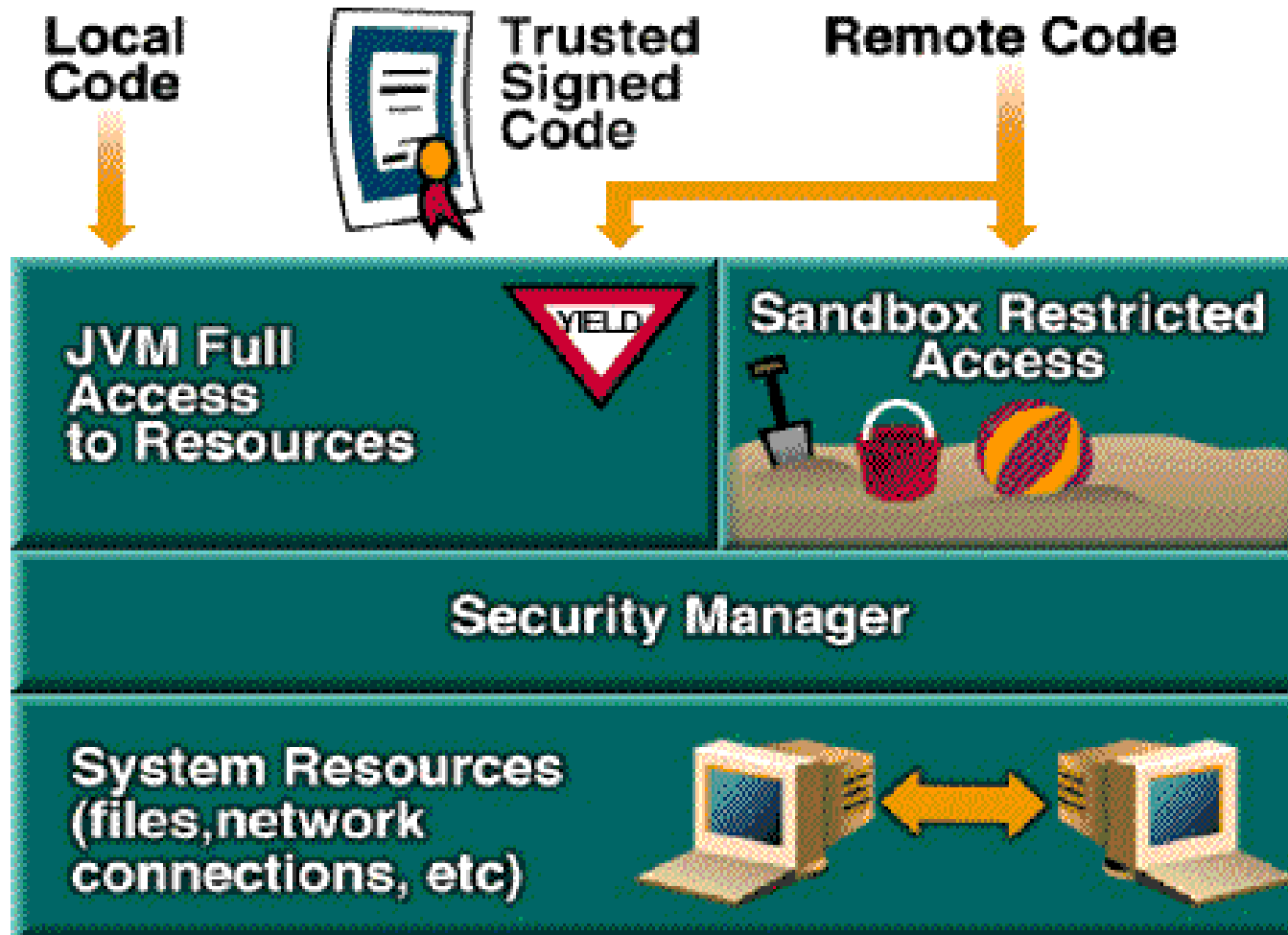
„Piaskownica” Java

- Tworzona przez Security Manager'a
- Uniemożliwia appletom między innymi
 - pisać, czytać i usuwać plików
 - uruchamiać innych programów
 - ładować biblioteki systemowe
 - otwierać połączenia sieciowe (za wyjątkiem hosta, z którego został pobrany)
- Tak restrykcyjna polityka bezpieczeństwa nie zawsze jest pożądana

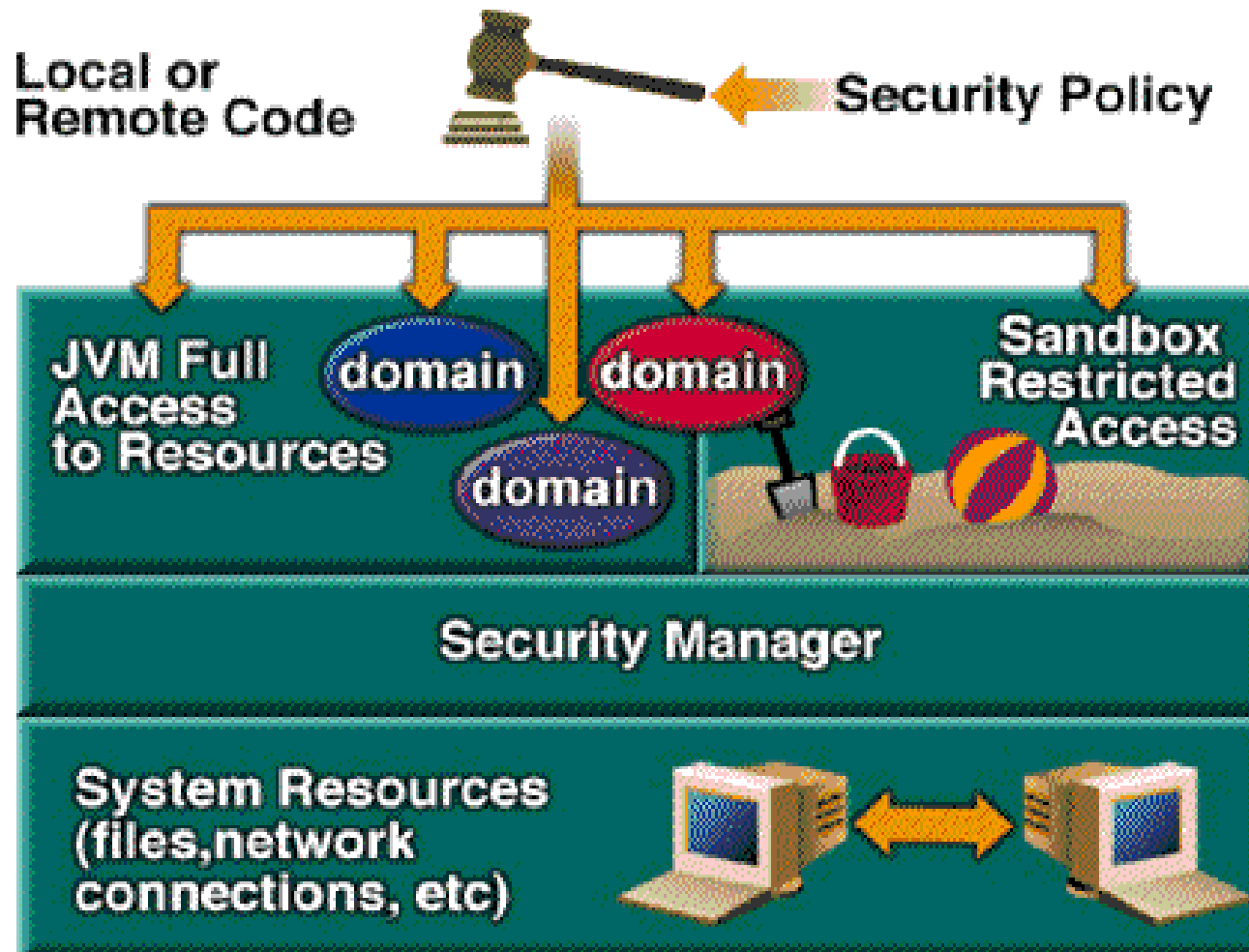
Java 1.0



Java 1.1



Java 1.2



Na koniec

- Problemy czekające na rozwiązanie
 - wywołania metod natywnych
 - zamknięte pakiety
 - problem niezgodności interfejsów
 -