CS1632, Static Analysis Part 1 Supplement: Type Checking

Wonsun Ahn

What is a Type?

1. A set of values. E.g.:
   - Integers between MIN_INT and MAX_INT
   - Strings with UNICODE characters
   - Objects with one integer and one string as member variables

2. A set of operations allowed on those values. E.g.:
   - Operators such as +, -, *, /, ==, !=, …
   - Method calls

- Two types may have same set of values but different allowed operations

Example of a buggy class

```
public class Pet {
  String name;
  Boolean isCat;
  public Pet(String name, Boolean isCat) { this.name = name; this.isCat = isCat; }
  public void meow() { System.out.println(name + " meows!"); }
  public void bark() { System.out.println(name + " barks!"); }
  public static void converse(Pet cat, Pet dog) {
    cat.meow();
    dog.bark();
  }
  public static void main(String[] args) {
    Pet dog = new Pet("Snoopy", false);
    Pet cat = new Pet("Garfield", true);
    converse(dog, cat);
  }
}
```

Bug may go undetected as code still runs

```
public class Pet {
  String name;
  Boolean isCat;
  public Pet(String name, Boolean isCat) { this.name = name; this.isCat = isCat; }
  public void meow() { System.out.println(name + " meows!"); }
  public void bark() { System.out.println(name + " barks!"); }
  public static void converse(Pet cat, Pet dog) {
    cat.meow();
    dog.bark();
  }
  public static void main(String[] args) {
    Pet dog = new Pet("Snoopy", false);
    Pet cat = new Pet("Garfield", true);
    converse(dog, cat);
  }
}
```

```
$ java Pet
Snoopy meows!
Garfield barks!
```

Better: Use runtime property checks

```
public class Pet {
  String name;
  Boolean isCat;
  public Pet(String name, Boolean isCat) { this.name = name; this.isCat = isCat; }
  public void meow() { assert isCat == true; System.out.println(name + " meows!"); }
  public void bark() { assert isCat == false; System.out.println(name + " barks!"); }
  public static void converse(Pet cat, Pet dog) {
    cat.meow();
    dog.bark();
  }
  public static void main(String[] args) {
    Pet dog = new Pet("Snoopy", false);
    Pet cat = new Pet("Garfield", true);
    converse(dog, cat);
  }
}
```

```
$ java Pet
Exception in thread "main"
java.lang.AssertionError
[Source line numbers]
```

Even Better: Use compile time type checks

```
public class Cat extends Pet {
  public Cat(String name) { super(name); }
  public void meow() { System.out.println(name + " meows!"); }
}

public class Dog extends Pet {
  public Dog(String name) { super(name); }
  public void bark() { System.out.println(name + " barks!"); }
}
```

- Created two types Cat and Dog that inherit from Pet

- Note that Cat and Dog have the same set of values, but different allowed operations
  => This is what we are going to leverage!

Even Better: Use compile time type checks

```
public class Pet {
  String name;
  public Pet(String name) { this.name = name; }
  public static void converse(Cat cat, Dog dog) {
    cat.meow();
    dog.bark();
  }
  public static void main(String[] args) {
    Dog dog = new Dog("Snoopy");
    Cat cat = new Cat("Garfield");
    converse(dog, cat);
  }
}
```

$ javac Pet.java Cat.java Dog.java
Error: Type mismatch error
[Source line numbers]

Type checks can be used to check any property

- Can check that only positive integers are passed to a method
  - By creating a “positive integer” type
- Can check that the program does not have data races
  - By creating a “thread local” type only accessible from one thread
- Can check that security guarantees are met
  - For example, that plaintext is never stored to the database

Property: plain text is never stored to the database

- This is correct code, but with no type checking for the property:

```
public class Text {
  String text;
  public Text(String s) { this.text = s; }
  public Text encrypt() { /* returns encrypted text */}
  public void storeDatabase() { /* stores text in database */ }
}
public class App {
  public static void main(String[] args) {
    Text text = new Text("Hello World");
    text = text.encrypt(); // Encrypt text before storing to DB
    text.storeDatabase();
  }
}
```

Property: plain text is never stored to the database

- With no type checking, incorrect code goes undetected:

```
public class Text {
  String text;
  public Text(String s) { this.text = s; }
  public Text encrypt() { /* returns encrypted text */}
  public void storeDatabase() { /* stores text in database */ }
}
public class App {
  public static void main(String[] args) {
    Text text = new Text("Hello World");
    // text = text.encrypt(); // Uh-oh. Commented out the encrypt.
    text.storeDatabase();     // App just stores the plain text. Fail!
  }
}
```

Property: plain text is never stored to the database

- This time, type checking is employed to check that property:

```
public class PlainText {
  String text;
  public PlainText(String s) { this.text = s; }
  public EncryptedText encrypt() { /* returns encrypted text */}
}
public class EncryptedText {
  String text;
  public void storeDatabase() { /* stores text in database */ }
}
public class App {
  public static void main(String[] args) {
    PlainText text = new PlainText("Hello World");
    EncryptedText text2 = text.encrypt();   // Encrypt text before storing to DB
    text2.storeDatabase();
  }
}
```

Property: plain text is never stored to the database

- This time, type checking is employed to check that property:

```
public class PlainText {
  String text;
  public PlainText(String s) { this.text = s; }
  public EncryptedText encrypt() { /* returns encrypted text */}
}
public class EncryptedText {
  String text;
  public void storeDatabase() { /* stores text in database */ }
}
public class App {
  public static void main(String[] args) {
    PlainText text = new PlainText("Hello World");
    // EncryptedText text2 = text.encrypt();   // Uh-oh. Forgot to encrypt.
    text.storeDatabase();  // Type error!  No storeDatabase method in PlainText
  }
}
```