Classes

Ch 10.1 - 10.3

class vs array

Arrays group together similar data types (any amount you want)

Classes (and structs) group together dissimilar types that are logically similar

class

A class is a new type that you create (much like int, double, ...)

Blueprint for all objects

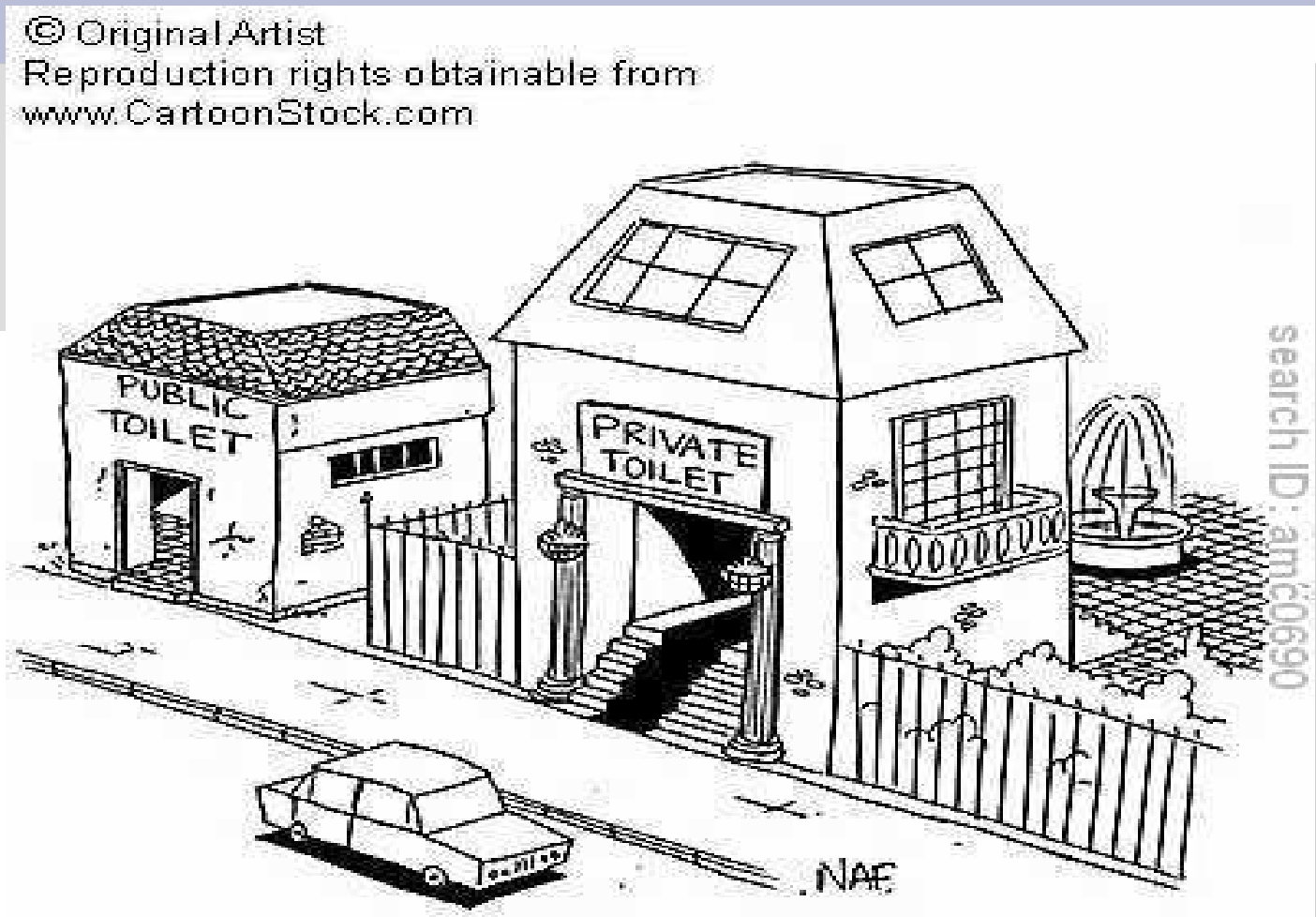
```
class date
{
public:
    int day;
    int month;
    int year;
    void print();
};
```

An instance of date class

```
int main ()
{
    int x;
    date today;
    date yesterday;
}
```

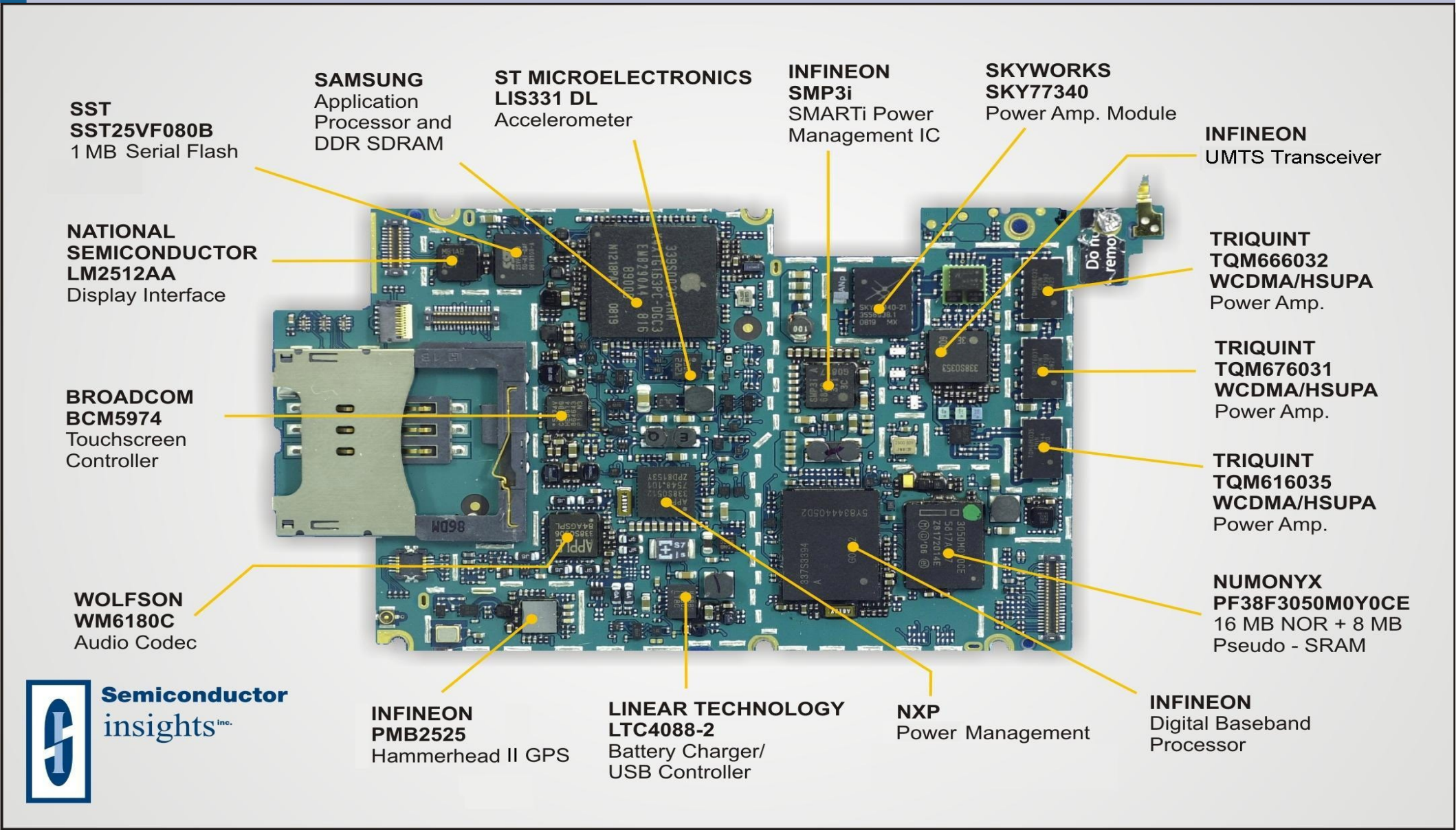
Another instance

public vs private




```
class date
{
private:
     int day;
     int month;
     int year;
public:
     void print();
     void setDate(int day, int month, int year);
};
```

public vs private



Semiconductor insights inc.

public vs private

Creating interfaces with public allows users to not worry about the private implementation

So... more work for you (programmer) less work for everyone else

public vs private

The public keyword allows anyone anywhere to access the variable/method

The private keyword only allows access by/in the class where the variable/method is defined
(i.e. only variables of this type can access this within itself)

public vs private

All variables should be private

While this means you need methods to set variables, users do not need to know how the class works

This allows an easier interface for the user (also easier to modify/update code)

(See: datePrivate.cpp)

public vs private

The idea is: if the stuff underneath changes, it will not effect how you use it

For example, you change from a normal engine to a hybrid engine... but you still fill it up the same way

public vs private

An important point: private just means only “date” things can modify the private variables of a “date” object

However, two different “date” objects can access each other's privates

(see: privateDates.cpp)

Constructors

The date class has two functions: setDate() and print()

As we need to run setDate() on a variable before it is useful anyways

In fact, such a thing exists and is called a constructor (run every time you create a variable)

Constructors

The class name and the constructor must be identical
(constructors also have no return type)

```
class date
{
private:
     int day;
     int month;
     int year;
public:
     date(int day, int month, int year);
     // ^^ constructor has same name as class
     void print();
};
```

(See: dateConstructor.cpp)

Constructors

If you don't put a constructor, C++ will make a default constructor for you (no arguments)

```
date();
```
← default constructor

```
date(int day, int month, int year);
```

To use the default constructor say this:

```
date never;
```
.... or ...
```
date never = date();
```

... not this:

```
date notWhatYouWant();
// ^ function declaration
```

Constructors

If you declared constructors you must use one of those

Only if you declare no constructors, does C++ make one for you (the default)

Note: our dateConstructor.cpp has no way to change the value of the date after it is created
(thus gives control over how to use class)

TL;DR Constructors

Constructors are functions, but with a few special properties:

(1) They have no return type
(2) They must have the same name as the class they are constructing
(3) If you want to make an instance of a class you MUST run a constructor
(and if you ever run a constructor, you are making an object)

#include

Just as writing very long main() functions can start to get confusing...

... writing very long .cpp files can also get confusing

Classes are a good way to split up code among different files

#include

You can #include your class back in at the top or link to it at compile time

You have to be careful as #include basically copies/pastes text for you

Will not compile if class declared twice (used in two different classes you #include)

#include

#include #include

date.cpp → date.hpp ← runDate.cpp

```cpp
date::date(int m, int d, int y)
{
    day = d;
    month = m;
    year = y;
}

void date::showDate()
{
    cout << month <<
    "/" << day <<
    "/" << year;
}
```

```cpp
class date
{
public:
    date(int m, int d, int y);
    void showDate();

private:
    int day;
    int month;
    int year;
};
```

```cpp
int main ()
{
    date today = date(3, 20, 2017);
    today.showDate();
}
```

Then compile with: g++ runDate.cpp date.cpp

#include

To get around this, you can use compiler commands in your file

"if not defined"

"define"

This ensures you only have declarations once (See: dateClass.hpp, dateClass.cpp, runDate.cpp)

```
#ifndef DATE
#define DATE
class date
{
public:
     int day;
     int month;
     int year;
     void print() const;
};
#endif
```