Tree Shaking Principles

When designing code please keep these principles in mind

Enums for features are not-tree-shakable

Here is an example of code which is not tree shakable.

```
export function query<T>(
    predicate: Type<any>| string[], descend?: boolean,
    read?: QueryReadType | Type<T>): QueryList<T> {
  ngDevMode && assertPreviousIsParent();
  const queryList = new QueryList<T>();
  const query = currentQuery || (currentQuery = new LQuery_());
  query.track(queryList, predicate, descend, read);
  return queryList;
}
```

Notice that `query()` takes the `QueryReadType` as enumeration.

```
function readFromNodeInjector(
    nodeInjector: LInjector, node: LNode, read: QueryReadType | Type<any>): any {
  if (read === QueryReadType.ElementRef) {
    return getOrCreateElementRef(nodeInjector);
  }
  if (read === QueryReadType.ViewContainerRef) {
    return getOrCreateContainerRef(nodeInjector);
  }
  if (read === QueryReadType.TemplateRef) {
    return getOrCreateTemplateRef(nodeInjector);
  }
  const matchingIdx = geIdxOfMatchingDirective(node, read);
  if (matchingIdx !== null) {
    return node.view.data[matchingIdx];
  }
  return null;
}
```

Sometimes later in the above code the `readFromNodeInjector` takes the `QueryReadType` enumeration and performs specific behavior.

The issue is that once the `query` instruction is pulled in it will pull in `ElementRef`, `ContainerRef`, and `TemplateRef` regardless if the `query` instruction queries for them.

A better way to do this is to encapsulate the work into an object or function which will then be passed into the `query` instead of the enumeration.

```
function queryElementRefFeature() {...}
```

```
function queryContainerRefFeature() {...}
function queryTemplateRefFeature() {...}

query(predicate, descend, queryElementRefFeature) {...}
```

this would allow the `readFromNodeInjector` to simply call the `read` function (or object) like so.

```
function readFromNodeInjector(
    nodeInjector: LInjector, node: LNode, readFn: (injector: Injector) => any) | Type<any>);
  if (isFeature(readFn)) {
    return readFn(nodeInjector);
  }
  const matchingIdx = geIdxOfMatchingDirective(node, readFn);
  if (matchingIdx !== null) {
    return node.view.data[matchingIdx];
  }
  return null;
}
```

This approach allows us to preserve the tree-shaking. In essence the if statement has moved from runtime (non-tree-shakable) to compile time (tree-shakable) position.